PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-010277
(43)Date of publication of application : 11.01.2002

(51)Int.Cl.

H04N 9/07
G06T 1/00
G06T 1/20
G06T 3/40
H04N 1/393
H04N 1/48
H04N 5/232
H04N 5/91
H04N 7/24
H04N 9/04
H04N 9/79
// H04N101:00

(21)Application number : 2000-404442
(22)Date of filing : 20.12.2000

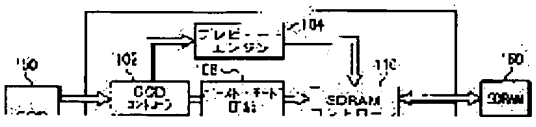
(71)Applicant : TEXAS INSTR INC <TI>
(72)Inventor : HUNG CHING-YU
TAMAMA HIDEO
OSAMOTO AKIRA
KOSHIBA OSAMU
YAMAUCHI AKIRA
ZHOU MINHUA
ILLGNER KLAUS
TALLURI RAJENDRA
YOO YOUNGJUN
LIANG JIE
TSAI MANDY
TSUNODA KIYOSHI
INAMORI SHINRI

(30)Priority

Priority number	Priority date	Priority country
1999 172780	20.12.1999	US
2000 214951	29.06.2000	
2000 215000	29.06.2000	US
2000 632543	04.08.2000	
		US
		US

(54) SYSTEM AND METHOD FOR DIGITAL STILL CAMERA

(57)Abstract:
PROBLEM TO BE SOLVED: To provide an architecture for a digital still camera having a characteristic selected from what integrates all camera peripheral devices.
SOLUTION: In the digital still camera(DSC), a preview engine, a burst mode compression/compression cancellation engine, an image pipeline, CCD+CCD



controller, and a memory + a memory controller are individually included. ARM micro-processor and DSP have control in common. A color filter array interpolation is executed by using red, blue and a green high frequency added for interpolation.

LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号
特開2002－10277
（P2002－10277A）
(43)公開日　平成14年１月11日(2002.1.11)

(51)Int.Cl.⁷		識別記号	FI		テーマコード(参考)	
Ｈ０４Ｎ	9/07		Ｈ０４Ｎ	9/07	Ａ	５Ｂ０５７
					Ｃ	５Ｃ０２２
Ｇ０６Ｔ	1/00	５１０	Ｇ０６Ｔ	1/00	５１０	５Ｃ０５３
	1/20			1/20	Ｂ	５Ｃ０５５
	3/40			3/40	Ｃ	５Ｃ０５９

審査請求　未請求　請求項の数11　ＯＬ　外国語出願　（全181頁）　最終頁に続く

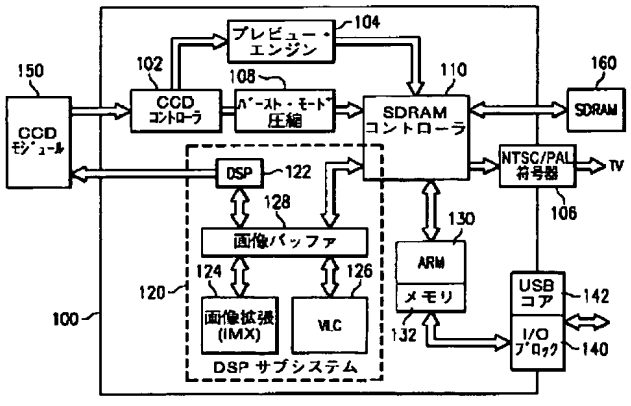
(21)出願番号　特願2000－404442(P2000－404442)
(22)出願日　平成12年12月20日(2000.12.20)
(31)優先権主張番号　１７２７８０
(32)優先日　平成11年12月20日(1999.12.20)
(33)優先権主張国　米国（ＵＳ）
(31)優先権主張番号　２１４９５１
(32)優先日　平成12年６月29日(2000.6.29)
(33)優先権主張国　米国（ＵＳ）
(31)優先権主張番号　２１５０００
(32)優先日　平成12年６月29日(2000.6.29)
(33)優先権主張国　米国（ＵＳ）

(71)出願人　590000879
テキサス　インスツルメンツ　インコーポレイテツド
アメリカ合衆国テキサス州ダラス，ノース　セントラルエクスプレスウエイ　13500
(72)発明者　チン　－　ユ　フン
アメリカ合衆国　テキサス、プラノ、　ボールドウィン　レーン　4633
(72)発明者　玉眞　秀雄
茨城県つくば市松代４－９－10
(74)代理人　100066692
弁理士　浅村　皓　（外３名）

最終頁に続く

(54)【発明の名称】　デジタルスチルカメラシステムおよび方法

(57)【要約】
【課題】　すべてのカメラ周辺機器を一体化したものから選択される特質を備えたデジタルスチルカメラのアーキテクチャを提供する。
【解決手段】　デジタルスチルカメラ（ＤＳＣ）に、プレビュー・エンジン，バーストモード圧縮／圧縮解除・エンジン，画像パイプライン，ＣＣＤ＋ＣＣＤコントローラおよびメモリ＋メモリ・コントローラが個別に含まれる。ＡＲＭマイクロプロセッサとＤＳＰとが制御を共有する。赤と青と補間に加えられた緑高周波とを用いることによるカラーフィルタアレイ補間を行う。



1

【特許請求の範囲】

【請求項１】　デジタルスチルカメラ用の集積回路であって、（ａ）制御機能を走らせるようプログラムされた第１のプログラム可能なプロセッサであって、ユーザインターフェース，メモリ用コントローラおよび画像獲得用コントローラに結合された第１のプロセッサと、

（ｂ）画像処理および圧縮機能を走らせるようプログラムされた第２のプログラム可能プロセッサであって、前記第１のプロセッサに結合された第２のプロセッサと、

（ｃ）前記第２のプロセッサに結合された第３のプロセッサであって、少なくとも４つの並列乗算および累算ユニットを含む第３のプロセッサと、を具備する、集積回路。

【請求項２】　デジタル画像のサイズを調整する方法であって、（ａ）Ｍ１×Ｍ２画素画像（Ｍ１およびＭ２は正の整数）を提供するステップと、（ｂ）画素位置に依存する一組のフィルタ係数で前記画像のＭ１画素の行をフィルタリングしてＭ１フィルタリング出力を生成するステップと、（ｃ）ステップ（ｂ）のフィルタリング出力のＭ１－Ｎ１を廃棄するステップと、（ｄ）Ｍ１×Ｍ２画像の他のすべての行についてステップ（ｂ）およびステップ（ｃ）を繰り返してＮ１×Ｍ２画像を生成するステップと、（ｅ）画素位置に依存する一組のフィルタ係数でＮ１×Ｍ２画像のＭ２画素の列をフィルタリングしてＭ２フィルタリング出力を生成するステップと、

（ｆ）ステップ（ｅ）のフィルタリング出力のＭ２－Ｎ２を廃棄するステップと、（ｇ）Ｎ１×Ｍ２画像の他のすべての行についてステップ（ｅ）およびステップ（ｆ）を繰り返してＮ１×Ｎ２画像を生成するステップと、を具備する、方法。

【請求項３】　カラーフィルタリングアレイ補間の方法であって、

第１のカラー・サブアレイを補間して第１のカラー・アレイを生成するステップと、

前記先行のステップからの前記第１のカラー・アレイをハイパスフィルタリングしてハイパス・第１のカラー・アレイを生成するステップと、

第２のカラー・サブアレイを補間して第２のカラー・アレイを生成するステップと、

前記ハイパス・第１のカラー・アレイを重み付けして前記第２のカラー・アレイに加算して修正・第２のカラー・アレイを生成するステップと、

第３のカラー・サブアレイを補間して第３のカラー・アレイを生成するステップと、

前記ハイパス・第１のカラー・アレイを重み付けして前記第３のカラー・アレイに加算して修正・第３のカラー・アレイを生成するステップと、

前記第１のカラー・アレイ，前記修正・第２のカラー・アレイおよび前記修正・第３のカラー・アレイで３色画像を画定するステップと、

2

を具備する、方法。

【請求項４】　画像を色調スケーリングする方法であって、（ａ）Ｍ×Ｎ画素画像を提供するステップであって、ＭおよびＮは正の整数であり、各画素が０からＫの範囲の強度を有する、ステップと、（ｂ）スケーリングされた累積分布関数Ｔ(Ｘ)を０からＫの範囲のＸについて決定するステップであって、０からＫの範囲の値で、Ｔ(Ｘ)は、Ｘ以下の強度を持つ画素の数を乗算されたのちにＭＮで割られたＫに等しい、ステップと、（ｃ）０から１の範囲のパラメータＡを選択するステップと、

（ｄ）ステップ（ａ）の画素の強度ＺをＡＴ(Ｚ)＋(１－Ａ)Ｚに調整するステップと、を具備する、方法。

【請求項５】　バイエル・カラーフィルタリングアレイ用の補間の方法であって、

緑サブアレイを補間して一時的緑アレイを形成するステップと、

該一時的緑アレイの補間された画素値をクランプし前記緑サブアレイの４つの隣接値のうちの２つの真中の値の範囲にあるようにして緑アレイを形成するステップと、

赤および青サブアレイを補間するステップと、

を具備する、方法。

【請求項６】　バイエル・カラーフィルタリングアレイ画像用の補間の方法であって、

緑サブアレイを補間して緑アレイを形成するステップと、

赤サブアレイを斜めに補間して第２の赤サブアレイを形成するステップと、

該第２の赤サブアレイを水平および垂直に補間して第２の赤アレイを形成するステップと、

青サブアレイを斜めに補間して第２の青サブアレイを形成するステップと、

該第２の青アレイを水平および垂直に補間して青アレイを形成するステップと、

を具備し、

前記斜めの補間は、前記緑アレイにおける対応画素の値による重み付けを含み、

前記水平および垂直の補間は、前記緑アレイにおける対応画素の値による重み付けを含み、

前記緑アレイ，前記赤アレイおよび前記青アレイがカラー画像を画定する、

方法。

【請求項７】　カラー画像センサ用のホワイトバランシングの方法であって、（ａ）白色光で照明されたカラー画像センサに出力緑，赤および青画素強度を提供するステップと、（ｂ）ステップ（ａ）の出力について緑画素強度の青画素強度に対する比を決定するステップと、

（ｃ）ステップ（ｂ）の比を乗算したものとステップ（ａ）における画素強度に依存するオフセットの加算との和として青画素強度を調整するステップと、（ｄ）ステップ（ａ）の出力について緑画素強度の赤画素強度に

対する比を決定するステップと、（ｃ）ステップ（ｂ）の比を乗算したものとステップ（ａ）における画素強度に依存するオフセットの加算との和として赤画素強度を調整するステップと、を具備する、方法。

【請求項８】　ビデオ復号および表示の方法であって、（ａ）符号化されたビデオフレームのシーケンス，ビデオフレーム境界時間のシーケンスおよびフレームの復号されたものについての蓄積位置を提供するステップと、（ｂ）前記シーケンスのｍ番目のフレームが復号されていると、TP_nを$Delta\ T$で割ったものとして決定されるｍ’で前記シーケンスのｍ’番目のフレームを復号するステップであって、$Delta\ T$がステップ（ａ）のビデオフレーム境界時間の連続するものの間の時間間隔であり、TP_nが、（ｉ）ｍ番目のフレームの表示時間TP_{n-1}と$Delta\ T$との和と（ｉｉ）ｍ番目のフレーム後に次に現れるべきフレームを復号するための開始時間T_sとフレームを復号するための見積り時間との和（該和は$Delta\ T$の倍数まで切り上げられている）とのうちの大きい方として決定される表示時間であり、開始時間T_sが、（ｉ）現在時間と（ｉｉ）復号されたｍ’番目のフレームが含まれる蓄積場所が表示されている先に復号されたフレームにより利用できるものとなる時間とのうちの大きな方である、ステップと、（ｃ）復号されたフレームを表示するステップと、（ｄ）インクリメントされたインデックスでステップ（ｂ）およびステップ（ｃ）を繰り返すステップと、を具備する、方法。

【請求項９】　補償カラーフィルタリングアレイ画像用の補間の方法であって、（ａ）第１のサブアレイ上のイエロー画素値Ｙｅ，第２のサブアレイ上のシアン画素値Ｃｙ，第３のサブアレイ上のマゼンタ画素値Ｍｇおよび第４のサブアレイ上の緑画素値Ｇを画素値の補償カラーフィルタリングアレイに提供するステップと、（ｂ）前記イエロー画素値を前記アレイ中のすべての画素に補間するステップと、（ｃ）前記シアン画素値を前記アレイ中のすべての画素に補間するステップと、（ｄ）前記マゼンタ画素値を前記アレイ中のすべての画素に補間するステップと、（ｅ）前記緑画素値を前記アレイ中のすべての画素に補間するステップと、（ｆ）ステップ（ｂ）からの画素イエロー値から量（Ｙｅ＋Ｃｙ－２＊Ｇ－Ｍｇ）／４を減算することによって画素イエロー値を調整するステップであって、Ｙｅがステップ（ｂ）からの画素イエロー値であり、Ｃｙがステップ（ｃ）からの画素シアン値であり、Ｍｇがステップ（ｄ）からの画素マゼンタ値であり、Ｇがステップ（ｅ）からの画素緑値である、ステップと、（ｇ）ステップ（ｃ）からの画素シアン値から量（Ｙｅ＋Ｃｙ－２＊Ｇ－Ｍｇ）／４を減算することによって画素シアン値を調整するステップであって、Ｙｅがステップ（ｂ）からの画素イエロー値であり、Ｃｙがステップ（ｃ）からの画素シアン値であり、Ｍｇがステップ（ｄ）からの画素マゼンタ値であり、Ｇがステップ（ｅ）からの画素緑値である、ステップと、（ｈ）ステップ（ｄ）からの画素マゼンタ値に量（Ｙｅ＋Ｃｙ－２＊Ｇ－Ｍｇ）／４を加算することによって画素マゼンタ値を調整するステップであって、Ｙｅがステップ（ｂ）からの画素イエロー値であり、Ｃｙがステップ（ｃ）からの画素シアン値であり、Ｍｇがステップ（ｄ）からの画素マゼンタ値であり、Ｇがステップ（ｅ）からの画素緑値である、ステップと、（ｉ）ステップ（ｅ）からの画素緑値に量（Ｙｅ＋Ｃｙ－２＊Ｇ－Ｍｇ）／４を加算することによって画素緑値を調整するステップであって、Ｙｅがステップ（ｂ）からの画素イエロー値であり、Ｃｙがステップ（ｃ）からの画素シアン値であり、Ｍｇがステップ（ｄ）からの画素マゼンタ値であり、Ｇがステップ（ｅ）からの画素緑値である、ステップと、を具備する、方法。

【請求項１０】　カラーフィルタリングアレイ画像用の補間の方法であって、（ａ）画素値のカラーフィルタリングアレイに、ｊおよびｋが偶数であるときにはｊ番目の行およびｋ番目の列に位置する画素用の赤画素値Ｒ（ｊ，ｋ）を、ｊおよびｋが奇数であるときにはｊ番目の行およびｋ番目の列に位置する画素用の青画素値Ｂ（ｊ，ｋ）を、ｊおよびｋの一方が偶数で他方が奇数であるときにはｊ番目の行およびｋ番目の列に位置する画素用の緑画素値Ｇ（ｊ，ｋ）を提供するステップと、（ｂ）画素値の行を水平に補間して、ｊが偶数であるときにはすべてのｋについて赤値Ｒ（ｊ，ｋ）および緑値Ｇ（ｊ，ｋ）の行を生成し、ｊが奇数であるときにはすべてのｋについて緑値Ｇ（ｊ，ｋ）および青値Ｂ（ｊ，ｋ）を生成するステップと、（ｃ）ｊが偶数であるときには、ステップ（ｂ）からの水平補間された赤行を垂直に補間して赤値Ｒ（ｊ－１，ｋ）を生成し、ｊが奇数であるときには、水平補間された青を垂直に補間して青値Ｂ（ｊ－１，ｋ）を生成するステップと、（ｄ）ステップ（ｂ）からの水平補間された緑画素行を垂直にフィルタリングして、フィルタリングされた緑出力値Ｇ”（ｊ，ｋ）を生成するステップと、（ｅ）緑値Ｇ（ｊ，ｋ）とフィルタリングされた緑値Ｇ”（ｊ，ｋ）との差値Ｄ（ｊ，ｋ）を定めるステップと、（ｆ）ｊが偶数であるときには、Ｄ（ｊ，ｋ）を赤値Ｒ（ｊ，ｋ）から減算して赤出力値Ｒ”（ｊ，ｋ）を生成し、ｍが奇数であるときには、Ｄ（ｊ，ｋ）を赤値Ｒ（ｊ，ｋ）に加算して赤出力値Ｒ”（ｊ，ｋ）を生成するステップと、（ｇ）ｍが偶数であるときには、Ｄ（ｊ，ｋ）を青値Ｂ（ｊ，ｋ）に加算して青出力値Ｂ”（ｊ，ｋ）を生成し、ｍが奇数であるときには、Ｄ（ｊ，ｋ）を青値Ｂ（ｊ，ｋ）から減算して青出力値Ｂ”（ｊ，ｋ）を生成するステップと、を具備する、方法。

【請求項１１】　再生バッファリングの方法であって、（ａ）第１および第２のバッファを備えるステップであって、これらのバッファがフル／非フル・指示器をそれ

ぞれ備え、位置指示器が前記第１および第２のバッファ内の位置を指示する、ステップと、（ｂ）前記第１のバッファについての前記フル／非フル・指示器が非フルを指示し、前記位置指示器が前記第２のバッファにおける位置を指示するときには、前記第１のバッファを符号化されたビットで充填するステップと、（ｃ）前記第２のバッファについての前記フル／非フル・指示器が非フルを指示し、前記位置指示器が前記第１のバッファにおける位置を指示するときには、前記第２のバッファを符号化されたビットで充填するステップと、（ｄ）前記位置指示器によって指示される位置で開始する前記第１および第２のバッファにおける１群のビットを復号するとともに、前記第１および第２のバッファを一つの巡回バッファを形成するように取り扱うステップと、（ｅ）ステップ（ｄ）によって取り除かれたビットに従って前記フル／非フル・指示器を更新するステップと、（ｆ）ステップ（ｂ）からステップ（ｅ）を繰り返すステップと、を具備する、方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、集積回路に関し、より詳しくは、デジタルカメラで用いる集積回路および方法に関する。

【０００２】

【発明が解決しようとする課題】近年、デジタルスチルカメラ（ＤＳＣ）は、非常に人気のある消費者用電気機器となっており、写真を趣味とする人，ウェブ製作者，不動産業者，保険の損害査定人，フォトジャーナリストからすべての写真愛好家に渡る広く多様なユーザにアピールしている。低電力デジタル信号プロセッサ（ＤＳＰ）の有用性と結びついた高解像度ＣＣＤアレイの近年の進歩により、伝統的なフィルムカメラによって提供される解像度および品質に極めて近いＤＳＣの開発が行われている。これらのＤＳＣは、データ蓄積，操作性及び伝送性の点から見て伝統的なフィルムカメラと比べてさらにいくつかの利点を提供している。撮られた画像をデジタル表示することにより、ユーザはその画像を任意の種類の電子媒体に簡単に取り込んだりそれらを任意の種類のネットワークを介して簡単に伝送することができる。撮られた画像を瞬時に見て選択的に蓄積する能力により、フィルムの浪費を最小限にするとともにその画像を再度撮る必要があるかどうかを瞬時に決定する柔軟性が提供される。デジタル表示することで、画像を撮った後にそれを訂正したり変更したり修正することができる。例えば、ベンカタラマンらの「次世代デジタルカメラの集積化及びソフトウェア開発の問題」，デジタル固体素子カメラ，設計と応用，３３０２会報，ＳＰＩＥ（１９９８年）を参照のこと。同様に、ＵＳＰ５，５２８，２９３およびＵＳＰ５，４１２，４２５には、メモリカードへの画像の蓄積およびバッテリーパワーカメラ用の節電を含むデジタルスチルカメラシステムの態様が開示されている。

【０００３】

【課題を解決するための手段】本発明は、プログラム可能カメラ機能＋短いオーディオ・ビデオ・クリップ能力とデュアルプロセッサ＋画像コプロセッサとバーストモード圧縮／圧縮解除エンジンとプログラム可能プレビュー・エンジンとＩｒＤＡ，ＵＳＢ，ＮＴＳＣ／ＰＡＬエンコーダ，ＲＧＢ用のＤＡＣ，ＵＡＲＴおよびコンパクトフラッシュカード／スマートメディアカード・インターフェースを含むすべてのカメラ周辺機器の一体化とから選択された特質を備えるデジタルスチルカメラ・アーキテクチャを提供する。これには、柔軟性，適応性および有効性を含む利点がある。

【０００４】

【発明の実施の形態】システムのあらまし

図１ａおよび図１ｂは、好ましい実施例のデジタルスチルカメラ（ＤＳＣ）およびシステムにおける様々な高レベル機能ブロックを示し、ここで、図１ｂは図１ａよりも詳細なものを提供している。特に、好ましい実施例の集積回路１００は、以下のアイテムを含む。すなわち、ＣＣＤまたはＣＭＯＳイメージャ１５０とインターフェースするＣＣＤコントローラ１０２と、ＮＴＳＣ符号器１０６またはデジタルＬＣＤインターフェースを用いてＣＣＤコントローラ１０２からのデータをディスプレイに適切なフォーマットに変換するプレビュー・エンジンブロック１０４と、損失のない（または、ユーザによって選択された損失のある）圧縮を用いてＣＣＤコントローラ１０２からの生画像データを圧縮するとともにその圧縮されたデータをＳＤＲＡＭコントローラ１１０を介して外部ＳＤＲＡＭ１６０に書き込むバースト・モード圧縮／圧縮解除エンジン１０８とである。その後、このデータは、ＤＳＰ１２２の制御のもとで圧縮解除エンジンによって圧縮解除され、処理され、表示されるかＳＤＲＡＭ１６０に再び記録される。ＤＳＰサブシステムブロック１２０（ＤＳＰ１２２およびｉＭＸ１２４と可変長符号器１２６およびバッファ１２８）は、捕獲モードにおいて画像データのすべての処理を行う。そのデータは、ＳＤＲＡＭコントローラ１１０への要求を通してＤＳＰ１２２によってＳＤＲＡＭ１６０から画像バッファ１２８にフェッチされ、また、ＤＳＰ１２２は、捕獲モードにおいて要求されるすべての画像処理および圧縮を行う。画像拡大プロセッサ（ｉＭＸ）１２４は、撮像アプリケーションのためにＤＳＰ１２２の性能を向上させるようＤＳＰ１２２の専用加速器として働く。

【０００５】ＲＩＳＣマイクロプロセッサ・サブシステム（ＡＲＭ１３０とメモリ１３２）は、インカメラ・オペレーティングシステム（ＯＳ）を支援する。ＶｘＷｏｒｋｓ，Ｍｉｃｒｏｉｔｒｏｎ，ＮｕｃｌｅｕｓおよびＰＳＯＳのような種々のＯＳおよび他のリアルタイム核

も回路１００で支援される。

【０００６】ＳＤＲＡＭコントローラ・ブロック１１０は、ＳＤＲＡＭ１６０とプロセッサ（ＡＲＭ１３０，ＤＳＰ１２２），ＣＣＤコントローラ１０２，ＴＶ符号器１０６，プレビュー・エンジン１０４などのようなすべての機能ブロックとの間の主要インターフェースとして働く。ＳＤＲＡＭコントローラ１１０は、８０ＭＨｚＳＤＲＡＭのタイミングまでを支援し、また、連続データアクセス用の低いオーバーヘッドを提供する。それはまた、ＣＣＤデータ入力およびＴＶディスプレイ・データ出力のリアルタイム・データ・ストリームを支援するアクセスユニットに優先順位を付ける能力を有する。

【０００７】カメラ・ショット・ツー・ショット遅延は、ＤＳＣエンジン１００がＣＣＤ１５０からデータを読み、それを処理し、それをＳＤＲＡＭ１６０に書き込むのにかかる時間である。その処理は、画像パイプライン段を含み、ＪＰＥＧ圧縮も含む。

【０００８】リアルタイムプレビューを支援するために、ＤＳＣエンジン１００は、ＣＣＤ１５０を「高速読出し」モードに設定し、データを処理し、データをＮＴＳＣフォーマットに変換し、組込みＬＣＤスクリーン（図１には示されていない）か場合によってはＴＶモニタ上にデータを表示する。

【０００９】オートフォーカス，オート露光およびオートホワイトバランス（３Ａ機能）が、ＤＳＣ１００がプレビュー・モードの動作状態にある間に、ＤＳＰ１２２によって行われる。ＤＳＰ１２２は、ＳＤＲＡＭ１６０から画像データを読み、リアルタイムで３Ａ機能を実行する。３Ａ機能用のアルゴリズムはプログラム可能である。

【００１０】インターレースおよびプログレッシブＣＣＤおよびＣＭＯＳイメージャ１５０の双方は、組込みＣＣＤ／ＣＭＯＳコントローラ１０２を用いてＤＳＣエンジン１００と直接連係する。

【００１１】Ｍｉｃｒｏｉｔｒｏｎのようなインカメラ・オペレーティングシステムは、ＤＳＣエンジン１００ではＡＲＭプロセッサ１３０で効率的に支援される。ＤＳＣエンジン１００はまた、「バースト・モード」の動作状態で高速シーケンスの画像を捕獲するのを支援する能力を有する。２メガピクセル画像の１０フレーム／秒までのバーストが支援される。バーストシーケンスの持続期間は、ＤＳＣシステムのＳＤＲＡＭ１６０のサイズによってのみ制限される。また、ＭＰＥＧ圧縮がショートクリップのために用いられてもよい。また、オーディオービデオの再生の能力は巡回バッファリングを含む。

【００１２】ＤＳＣ回路１００はまた、プログラミングおよびＡＲＭ１３０での中断処理用のＵＳＢコア１４２を備えるＩ／Ｏブロック１４０を含む。

【００１３】ＣＣＤモジュール１５０は、画像を検知するＣＣＤイメージャと、必要な信号がＣＣＤをクロックするためのドライバ電子回路およびタイミング発生器と、相関二重サンプリングおよび自動利得制御電子回路とを含む。その後、このＣＣＤデータが、デジタル化され、ＤＳＣエンジン１００に送り込まれる。

【００１４】ＳＤＲＡＭ１６０は、任意の便宜的および高速ＳＤＲＡＭであってよい。

【００１５】ＤＳＣシステムは、画像にテキスト／音声で注釈を付ける能力もあって更により用途が多い。好ましい実施例のプログラム可能ＤＳＰにより、モデムおよび／またはインターネットへの直接接続用のＴＣＰ／ＩＰインターフェースを簡単に含むことが可能となる。ＤＳＣは、複雑なマルチタスキング・オペレーティングシステムを走らせて種々のリアルタイム・タスクをスケジュールする。

【００１６】このように、好ましい実施例は、プログラム可能カメラ機能と二重プロセッサ（ＡＲＭおよびＤＳＰ）および画像コンプレッサとバースト・モード圧縮／圧縮解除エンジンとプログラム可能プレビュー・エンジンとＩｒＤＡ，ＵＳＢ，ＮＴＳＣ／ＰＡＬ符号器，ＲＧＢ用のＤＡＣ，ＵＡＲＴおよびコンパクトフラッシュカード／スマートメディアカード・インターフェースを含むすべてのカメラ周辺器の一体化のためのプラットフォームを提供する。また、プラットフォームは、同じ集積回路上にカメラ機能およびデジタルオーディオ再生の双方を提供することができる。

【００１７】以下のセクションは機能およびモジュールの更なる詳細を提供する。

【００１８】ＤＳＣ動作モード

好ましい実施例のシステムは、以下の通り、動作の（１）プレビュー・モード，（２）捕獲モード，（３）再生モードおよび（４）バースト・モードを有する。

【００１９】（１）プレビュー・モードは、図２に例示されるようなデータ・フローを有する。ＡＲＭ１３０は、ＣＣＤ１５０を高フレーム速度読出しモード（垂直解像度が低減される）に設定する。ＡＲＭ１３０は、プレビュー・エンジン１０４をイネーブルするとともに、デフォルト・パラメータのために適切なレジスタを設定する。生ＣＣＤデータは、プレビュー・エンジン１０４に流れ込み、プレビュー・エンジン処理の後にＳＤＲＡＭ１６０に流れ込む。ＡＲＭ１３０は、ＴＶ符号器１０６をイネーブルしてプレビュー・エンジン出力を表示させる。プレビュー・エンジン１０４の処理（ハードウェア）は、利得制御と、ホワイトバランスと、ＣＦＡ補間と、ダウンサンプリングと、ガンマ補正と、ＲＧＢからＹＵＶへの変換とを含む。ＡＲＭ１３０は、必要なときはいつでも、オート露光およびオートホワイトバランスを行うようにＤＳＰ１２２に指令する。ＤＳＰ１２２の処理は、オート露光とオートホワイトバランスとオートフォーカスとを含む。ＡＲＭ１３０は、プレビュー・エンジン１０４のための新しいパラメータを受け取り、こ

れらのパラメータをプレビュー・エンジン・ハードウェアにローディングする。出力は、完全解像度ＣＣＩＲ６０１ＮＴＳＣ／ＰＡＬと、利得，ホワイトバランスおよびオートフォーカスのリアルタイム更新とである。

【００２０】（２）捕獲モードは、図３ａに例示されるようなデータ・フローを有する。ＡＲＭ１３０は、ＣＣＤ１５０を「微細」読出しモード、最大限解像度に設定する。ＣＣＤデータは、ＳＤＲＡＭコントローラ１１０を通してＳＤＲＡＭ１６０に直接読み込まれる。ＡＲＭ１３０は、ＤＳＰ１２２（とｉＭＸ１２４およびＶＬＣエンジン１２６）に捕獲処理（ブラック・クランプと、誤りピクセル訂正と、陰影補償と、ホワイトバランシングと、ガンマ補正と、ＣＦＡ補間と、色空間変換と、エッジ・エンハンスメントと、誤色抑制と、４：２：０ダウンサンプリングと、ＪＰＥＧ圧縮）を行うように指令する。ＤＳＰは、圧縮データをＳＤＲＡＭに記憶する。ＡＲＭ１３０は、圧縮データをコンパクトフラッシュ／スマートメディア１８２に書き込む。

【００２１】計算は、２つのスレッドとしてスケジュールされる。すなわち、１つのスレッドでのｉＭＸと、他方のスレッドでの他のユニットとである。図３ｂは、バッファＡおよびバッファＢに関係するスレッドでのタイミングおよびデータ・フローを示す。

【００２２】（３）再生モードは、図４に例示されるようなデータ・フローを有する。ＡＲＭ１３０は、ＣＦＣ／スマートメディア１８２からの圧縮データをＤＭＡ１６２を用いてＳＤＲＡＭコントローラ１１０を介してＳＤＲＡＭ１６０に読み込む。ＡＲＭは、「再生」を行うようにＤＳＰ１２２に指令する。ＤＳＰ処理（ＤＳＰ１２２とｉＭＸ１２４およびＶＬＣエンジン１２６）は、ＪＰＥＧ復号（ビットストリーム構文解析，ＩＤＣＴ，ＶＬＤおよびアスペクト比用のダウンサンプリング）を含み、非圧縮画像データをＳＤＲＡＭに記憶する。ＡＲＭは、ＴＶ符号器１０６をイネーブルして画像をＴＶ／ＬＣＤディスプレイに表示する。オーディオおよびビデオ（例えば、ＭＰＥＧ圧縮された）クリップが再生されてもよいということに留意願いたい。

【００２３】（４）バースト捕獲モードは、図５に例示されるようなデータ・フローを有し、また、図６は、オフラインデータ処理を示す。ＡＲＭ１３０は、ＣＣＤ１５０を微細解像モードに設定する。ＡＲＭは、バースト圧縮パラメータ，バースト長，フレーム数／秒，圧縮比（損失のある、損失のない）などを設定する。ＡＲＭは、バースト圧縮エンジン１０８をイネーブルして生ＣＣＤデータをＳＤＲＡＭ１６０に書き込む。ＡＲＭは、記憶された生ＣＣＤ画像の各々をバーストにおいて処理するようにＤＳＰに信号を送る。バースト・モード圧縮解除エンジン１０８は、バースト捕獲された画像の各々を圧縮解除する。ＤＳＰは、通常の捕獲にある画像の各々を処理するとともに、ＪＰＥＧビットストリームをＳＤＲＡＭ１６０に書き込む。

【００２４】バースト捕獲モードは、毎回ＡＲＭ１３０による異なるＪＰＥＧビットストリームでの通常の再生ルーティンへのコールを繰り返すことによって、達成される。

【００２５】好ましい実施例はまた、ＭＰＥＧ１捕獲モードと再生モードとを有する。

【００２６】画像獲得

ＤＳＣは、通常は、高品質画像が記憶される前に多数の処理ステップを行わなければならない。最初のステップは画像獲得である。場面から反射された強度分布が、光システムによってイメージャにマッピングされる。好ましい実施例はＣＣＤを用いるが、ＣＭＯＳへの移行が画像処理原理を変えるものではない。カラー画像を提供するために、イメージャ（ＣＣＤまたはＣＭＯＳ）は、各画素を（各ＣＣＤフォトサイト上の蒸着された染料のような）カラーフィルタによってマスクさせる。この生画像データは、通常は、カラー・フィルタ・アレイ（ＣＦＡ）と呼ばれる。ＣＣＤにおける画素のアレイのマスキングパターンとフィルタカラー原色とは、異なる製造業者間で異なっている。ＤＳＣアプリケーションでは、最も普通に用いられているＣＦＡパターンは、ＣＣＤアレイ全体に渡って傾いている２×２セル素子からなるＲＧＢバイエル（Ｂａｙｅｒ）パターンである。図７ａは、ＣＣＤカメラに続く行列ブロックにおけるこのバイエル・パターンの部分集合を示す。画素の半分は緑に対して感受性があることと、赤と青とは緑とバランスがとられていることとに留意すべきである。図７ｂは、黄，シアン，緑およびマゼンタ画素を備える代替の補完カラーＣＦＡパターンの部分集合を示す。

【００２７】画像パイプライン

画像が最終的に圧縮または表示用の使用可能なフォーマットで提供される前に、ＣＦＡデータは著しい量の画像処理を受ける必要がある。これらの処理段のすべては、まとめて「画像パイプライン」と呼ばれる。好ましい実施例のＤＳＣは、高品質の画像が記憶される前に、多数の処理ステップを行う。画像パイプライン処理タスクのほとんどは、乗算／累算（ＭＡＣ）集中動作であり、ＤＳＰを好ましいプラットフォームにする。様々な画像パイプライン処理段が以下のセクションで記述される。

【００２８】Ａ／Ｄコンバータ

ＣＣＤイメージャ・データをデジタル化するＡ／Ｄコンバータは、１０～１２ビットの解像度を有する。これによって、入力画像値を表すのに良好なダイナミックレンジが可能となる。もちろん、より高い解像度は、より高い品質の画像を意味するが、より多くの計算およびより遅い処理を意味する。また、より低い解像度はその反対を意味する。Ａ／ＤコンバータはＣＣＤモジュールの一部であってもよい。

【００２９】ブラック・クランプ

Ａ／Ｄ変換の後、「ブラック」画素は、これらの画素位置である電流（電荷蓄積）をまだ記録するＣＣＤのために、必ずしも０値を有するわけではない。ＣＣＤイメージャによって表される画素値のダイナミックレンジを最適化するために、ブラックを表す画素は０値を有さなければならない。ブラック・クランプ機能は、各画素値からオフセットを減算することによって、このための調整を行う。処理のこの段階で画素ごとに１つのカラー・チャンネルのみがあることに留意願いたい。

【００３０】誤り画素補間

ＣＣＤアレイ（とりわけ５００，０００を超える素子を備えるアレイ）は、欠陥のある（失われている）画素を有しているかもしれない。その失われている画素値は簡単な補間によって充填される。補間もＣＦＡ補間段で行われるため、高いオーダの補間が必要なわけではない。したがって、この予備補間ステップを行う主な理由は、失われているデータを削除することによって画像処理を通常のものにすることである。

【００３１】典型的には、失われている画素の位置はＣＣＤ製造業者から得られる。誤り画素位置はまた、ＤＳＣエンジンによってオフラインで計算できる。例えば、カメラ初期化動作の間、レンズのキャップが閉められた画像が捕獲される。誤り画素は「白いスポット」として現れ、一方、その画像の残りは黒である。それにより、誤り画素位置は、簡単な閾値検出器で識別でき、ビット・マップとしてメモリに記憶できる。

【００３２】ＤＳＣの通常動作の間、誤り画素位置の画像値は、単純な双線形補間技術によって充填される。

【００３３】レンズ歪補償

レンズの不完全性によって導入される非線形性のために、画像の明るさは画像の中心から画像の境界にかけて低減する。これらのレンズ歪の影響は、それの空間的な位置の機能として各画素の明るさを調整することによって補償される。レンズ歪を記述するパラメータは、最終システムで測定され、レンズ製造業者によって供給される情報によって支援される必要がある。

【００３４】レンズ調整は、画素強度を定数と乗算することによって達成することができ、その定数の値は画素位置で異なる。その調整は、水平方向および垂直方向の双方について行われる必要がある。

【００３５】ホワイトバランス

ホワイトバランシングは、ある光の状態のもとで検知された３刺激値を、表示されるとするならば白が再び白として現れるように変換を試みるものである。一般に、カメラによって捕獲される色は、場面を捕獲するときにそれらが見られた通りに出力デバイス上に現れることはない。それには２，３の理由が考えられる。第１に、スペクトラム範囲にわたってカラーフィルタの感度がわずかに異なる。完全な白色光源（一定の光スペクトラム）で露光されると、ＣＣＤによって検知される３刺激値はわずかに異なる。第２に、ＣＣＤモジュール全体および光システムの設計は３刺激値の不均衡に加わる。第３に、場面を記録する間に存在する典型的な照明体が一定ではない。照明体はある「カラー」を有しており、それは、典型的には、「カラー温度」（または、相関カラー温度）として特徴付けられる。照明体１のもとで捕獲された画像が異なる照明体のもとで表示されると、カラーの見た目が変わる。これによって、ホワイト領域が少し赤にまたは少し青に変えられる。

【００３６】ホワイトバランスのためにいくつかの異なるアプローチが知られている。それらのほとんどは、ホワイトパッチについての結果として得られる３刺激値が特定の値を有するように、赤および青チャンネルをあるファクターと乗算する。

【００３７】

【数１】

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} a1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & a2 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

中間（灰色）パッチについてはR'＝G'＝B'

【００３８】しかしながら、後で説明されるように、このアプローチでは照明体の変化についての補正が得られない。したがって、好ましい実施例のシステムにおけるホワイトバランシングの実行は、センサ・モジュールの不均衡を補正する。照明体補正は、カラー補正セクションにおける後の段で取り扱われる。

【００３９】利得ファクターを計算する典型的な技術は、

【００４０】

【数２】

（１）等価エネルギー

$$a1 = \Sigma_{(x,y)} g^2(x, y) / \Sigma_{(x,y)} r^2(x, y)$$

（２）灰色ワールド仮定

$$a1 = \Sigma_{(x,y)} g(x, y) / \Sigma_{(x,y)} r(x, y)$$

（３）画像における最大値がホワイト

$$a1 = max_{(x,y)} g(x, y) / max_{(x,y)} r(x, y)$$

【００４１】それらのすべてがあらゆる場合において成り立つわけではない。したがって、ホワイトバランシングを主にイメージャ・モジュール特性の補正として定義することによって、補正値を得るためのアルゴリズムはほとんど場面から独立させることができる。

【００４２】図８は、プレビュー・エンジンを簡素化して実現したものを示しており、ＣＣＤセンサが線形範囲で作動する限り良好な結果が得られている。以下のホワイトバランス・セクションは、より洗練された方法を記述する。

【００４３】ガンマ補正

画像を表示するのに用いられるディスプレイ装置（ＴＶモニタ）および画像を印刷するのに用いられるプリンタは、画像灰色値と実際に表示される画素強度との間の非線形マッピングを有する。それ故に、好ましい実施例のＤＳＣにおいては、ガンマ補正段は、ＣＣＤ画像を補償してそれらを最終的な表示／印刷のために調整する。

【００４４】ガンマ補正は非線形動作である。好ましい実施例は、その補正をテーブル・ルックアップとして実行する。テーブル・ルックアップの利点は、高速および高柔軟性である。ルックアップ・テーブル・データはカメラ製造業者によって提供されるものであってもよい。

【００４５】１２ビットのデータでは、最大限ルックアップ・テーブルは、各エントリが８～１２ビットの４Ｋエントリを有する。より小さなルックアップ・テーブルについては、補正曲線への区分線形近似（piecewise linear approximation）を用いることができる。例えば、６つの最も重要なビットは、エントリがベース値（８～１２ビット）とスロープ（６ビット）との値の複数のペアである６４エントリ・ルックアップ・テーブルにアドレスすることができる。そして、６つの最も重要ではないビットとスロープとの積がベース値に加算されて、８～１２ビットの最終的に補正された値を生成する。図９ｂは区分線形近似曲線を例示し、図９ｃは対応する動作を例示する。

【００４６】ＬＣＤディスプレイは線形であると考えることができ、ガンマ補償を不要とすることに留意願いたい。しかしながら、ＬＣＤディスプレイ・モジュールは、通常は、ＮＴＳＣ入力（それはすでにガンマ補償されている）を期待しており、それ故に、この期待されたガンマ補正を補償するために何らかの「ガンマ不補正」（逆ガンマ補正）を行う。それで、そのようなＬＣＤプレビュー・モジュールを用いる好ましい実施例のＤＳＣにおいては、ガンマ補正をなおも行っていて、ＮＴＳＣは信号をＬＣＤモジュールに送出する前にそれを符号化する。

【００４７】ガンマ補正は、画像パイプライン処理の全段の終わりにあって、ディスプレイに行く直前に行われる。代わりに、画像パイプラインは、パイプラインにおいてより早くすなわちＣＦＡ補間段の前にガンマ補正を行うことができる。

【００４８】ＣＦＡ補間

カラーフィルタ・アレイ（ＣＦＡ）を用いることにより、各カラー平面の有効解像度が低減される。任意の所定の画素位置には、１つのカラー画素情報（ＲＧＢ原色の場合にはＲかＧかＢ）のみがある。しかしながら、ＤＳＣにおける各画素では最大限カラー解像度（Ｒ，ＧおよびＢ）を生成することが要求される。これをすることができるために、失われている画素値（Ｇ位置でのＲおよびＢなど）がＣＦＡ補間において局所的な隣接部の値から補間によって再構築される。このシステムにおけるＤＳＰの利点を生かすために、補間フィルタとしてＦＩＲ核（FIR-kernel）が使用されている。フィルタの長さと重みとは実例によって異なる。帯域間の関係もまた考慮されなければならない。図１０は、ハードワイヤード・プレビュー・エンジン・モジュールにおけるＣＦＡ補間の実現を説明する。それは、基本的には、垂直補間が続く水平補間用の１Ｄ・ＦＩＲ核を使用する。

【００４９】高品質画像処理用のＤＳＰサブシステムの実行は、それが完全にプログラム可能であって２Ｄフィルタ核を使用することができるという点で異なっている。改良されたＣＦＡ補間技術用のいくつかの背景情報と提案とが、以下に続くセクションで示される。

【００５０】カラー補正

捕獲と再生／印刷とで異なる照明体によって引き起こされるカラー外観の変化は、赤，緑および青チャンネルを独立してバランスさせることだけでは補正できない。これを補償するために、色調（カラー）補正行列は、照明体を考慮に入れた補正ＲＧＢ画素値にＲＧＢ画素値をマッピングする。

【００５１】その原理は以下の通りである。Ｉ１は記録照明体を示すＮ×Ｎ斜行列を示し、Ｓは各カラーについ

て１列ベクトルを備えるイメージャ・モジュールのスペクトラム特性を示すＮ×３行列を示し、Ｒは場面の反射率を示す１×Ｎ列ベクトルを示すものとする。画素位置での測定された３刺激値Ｘ１は、次式で与えられる。

【００５２】

【数３】$X1^{T}=R^{T}*I1*S$

ここで、

$SS=S*S^{T}$

【００５３】場面がＩ２によって照明されているならば測定されたであろうＸ２に測定された３刺激値Ｘ１を変換することができる。

【００５４】

【数４】$X2^{T}=X1^{T}*S^{T}*SS^{-1}*I1^{-1}*I2*S$

【００５５】３×３変換行列$S^{T}*SS^{-1}*I1^{-1}*I2*S$は、センサのスペクトラム応答が測定されることができるものとして、オフラインで計算することができる。このように、異なる照明体についての一組のカラー補正行列をカメラに記憶すれば十分である。カラー外観の主観的好みはユーザ間で変わるので、これらをカラー補正行列に含めるか画像処理パイプラインに別のステップを加える（例えば、「色調スケール」）ことは容易に可能である。

【００５６】カラー空間変換

ＣＦＡ補間およびカラー補正の後、画素は、典型的には、ＲＧＢカラー空間にある。圧縮アルゴリズム（ＪＰＥＧ）はＹＣｂＣｒカラー空間に基づいているので、カラー空間変換が実行されなければならない。好ましい実施例のＤＳＣはまた、ＴＶでの表示のために、また、ＬＣＤプレビューに送出するために、ＮＴＳＣ信号出力を生成する。それ故に、ＲＧＢからＹＣｂＣｒへのカラー空間変換が実行される必要がある。これは線形変換であり、Ｙ値，Ｃｂ値およびＣｒ値はそれぞれ、そのピクセル位置でのＲ値，Ｇ値およびＢ値の重付け合計である。図１１ａは、ハードワイヤード・プレビュー・エンジンにおいて実現されるカラー変換を例示する。ＤＳＰ（再生）実行は、原理的には同様であるが、より高い精度の変換を可能とする。

【００５７】

【数５】

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} \alpha 1 & \alpha 2 & \alpha 3 \\ \alpha 4 & \alpha 5 & \alpha 6 \\ \alpha 7 & \alpha 8 & \alpha 9 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

【００５８】エッジ・エンハンスメント

ＣＦＡ補間の後に、画像は補間フィルタのローパスフィルタリング効果により少し「スムーズ」に見える。画像をシャープにするためには、Ｙ成分のみについて動作すれば十分である。各画素位置では、典型的には二次元ＦＩＲフィルタであるエッジ検出器を用いてエッジの大きさを計算する。好ましい実施例は３×３ラプラス演算子を用いる。エッジの大きさは、画像のシャープさを高めるために原輝度（Ｙ）画像に加算される前に、閾値がとられてスケーリングされる。エッジ強調はハイパスフィルタであり、このハイパスフィルタもノイズを増幅する。この増幅されたノイズを回避するために、閾値機構が用いられてエッジ上にある画像のこれらの部分のみを強調する。増幅されたエッジの大きさは変えられる。閾値動作がノイズの増幅を低減するのに必要である。したがって、エッジの素子であるこれらの画素のみが強調される。輝度チャンネルに加算される強調信号は図１１ｂのようにグラフィックに表すことができ、パラメータｔ１，ｔ２およびスロープｓ１は最高品質を得るのに必要であると思われるように選択されることができる。

【００５９】誤りカラー抑制

エッジ強調はＹ画像において行われるのみであることに留意願いたい。エッジでは、カラー・チャンネルの補間画像が十分に整列されないかもしれない。これは、シャープなエッジで厄介な虹のような人工物を引き起こす。したがって、Ｙ成分におけるエッジでカラー成分Ｃｂ，Ｃｒを抑制することによって、これらの人工物を低減することができる。エッジ検出器の出力に基づいて、カラー成分Ｃｂ，Ｃｒは、画素当りベースで１未満の係数と乗算され、誤りカラー人工物を抑制する。

【００６０】画像圧縮

画像圧縮ステップは、典型的には、約１０：１から１５：１に画像を圧縮する。好ましい実施例のＤＳＣはＪＰＥＧ圧縮を用いる。これは、良好な性能を与えるＤＳＴベース画像圧縮技術である。

【００６１】オート露光

場面の明るさが変化することにより、良好な全体画像品質を得るのに、ＣＣＤの露光を制御してデジタル化画像のダイナミックレンジを最大にすることが必要である。露光制御の主なタスクは、シャッタースピードおよび可能ならば光学システムの開口を制御することによって、センサを線形レンジで作動させ続けるものである。虹彩絞りを閉じることとシャッター速度を遅くすることとはお互いに補償し合うので、露光が変わらないままである一定のパラメータ範囲が存在する。高速で動く場面を捕獲するというような他の制約がユーザによって望まれるときにある程度までしかこれを達成することができないのは明白である。センサを線形レンジで作動させ続けようとすることに加えて、ＡＤＣのダイナミックレンジしたがってデジタル化画像を最大にすることが望まれる。これは、ＡＦＥにおけるＰＧＡを制御することによって行われる。関連する制御パラメータを得るのに必要な処理は、ＤＳＰで行われる。

【００６２】オートフォーカス

画像処理を通してＤＳＣにおけるレンズ焦点を自動的に調整することもできる。オート露光と同様に、これらの

オートフォーカス機構もフィードバックループにおいて作動する。それらは、レンズ焦点の品質を検出するために画像処理を行って、画像がシャープに焦点を結ぶまで反復してレンズ・モータを動かす。オートフォーカスは、先に説明されたエッジ強調からのエッジ測定に依存する。

【００６３】再生

好ましい実施例のＤＳＣはまた、ユーザが捕獲画像をカメラのＬＣＤスクリーン上か外部ＴＶモニタ上で見ることができるようにする。捕獲画像はＪＰＥＧビットストリームとしてＳＤＲＡＭ（または、コンパクトフラッシュメモリ）に記憶されるので、再生モード・ソフトウェアもＤＳＰ上に備えられる。この再生モード・ソフトウェアは、ＪＰＥＧビットストリームを復号し、復号された画像を適当な空間解像度に縮小拡大し、それをＬＣＤスクリーンおよび／または外部ＴＶモニタ上に表示する。

【００６４】ダウンサンプリング

好ましい実施例のＤＳＣシステムでは、ＪＰＥＧデータを復号した後の再生モードの間の画像は、ＣＣＤセンサの解像度、例えば２メガピクセル（１６００×１２００）である。この画像は、ＣＣＤセンサの解像度によってより大きなものとすることもできる。しかしながら、表示する目的のためには、この復号データは、ＮＴＳＣ符号器に送出することができるようになる前に、ＮＴＳＣ解像度（７２０×４８０）にダウンサンプルされなければならない。それ故に、ＤＳＣは、再生モードの末尾でダウンサンプリング・フィルタを実現し、それによって、更なるＤＳＰ計算を要求する。好ましい実施例は、この更なるＤＳＰ計算の問題を、ＪＰＥＧ圧縮解除モジュールの一部として含まれるＤＣＴドメイン・ダウンサンプリング方式によって解決する。ＪＰＥＧ圧縮解除は本質的に３つの段（すなわち、最初のエントロピー復号化段，それに続く反量子化段および最後のＩＤＣＴ段）を含むことに留意願いたい。ＪＰＥＧでは、ＩＤＣＴは８×８画素のブロックで行われる。好ましい実施例は、ＩＤＣＴドメインにおいて４×４ＩＤＣＴを（８×８ＤＣＴ係数ブロックからの）左上４×４ＤＣＴ係数まで使用することによって、２メガピクセル画像をＮＴＳＣ解像度（４／８ダウンサンプリング）にダウンサンプルし、これによって、ＩＤＣＴおよび４／８ダウンサンプリングの双方を１つのステップで効率的に達成する。サンプリング比は、１／８（最小画像）と８／８（最大解像度画像）との間で変えることができる。

【００６５】分離可能二次元４ポイントＩＤＣＴが適用されて、左上（低空間周波数）４×４ＤＣＴ係数から画像画素の４×４ブロックを得る。この低オーダーＩＤＣＴによって、反折返し（anti-aliasing）フィルタリングと８から４のデシメーション（decimation）とを効率的に結合する。使用された反折返しフィルタは、保持されたＤＣＴ係数をスケーリングすることなくＤＣＴドメインにおいて１６の最低周波数成分のみを保持するという簡単な動作に対応する。この簡単なフィルタは折返し（aliasing）効果を低減するのに有効であるが、好ましい実施例は、更に折返しを低減するために、より良好な周波数応答を備えるローパスフィルタを有する。他のローパスフィルタの使用は、スケーリング係数が各ＤＣＴ係数の位置である保持された係数のスケーリングを導く。ＤＣＴドメイン・ダウンサンプリング技術は計算の複雑さを増大しないことに留意願いたい。実際、それは、エントロピー復号化後のＪＰＥＧ復号化段が左上４×４係数を除いた８×８ＤＣＴ係数全体を取り扱う必要はないので、計算を低減する。他の反折返しフィルタの使用は、係数スケーリング動作が低オーダーＩＤＣＴ動作にマージされることができるので、複雑さを加えない。このＤＣＴドメイン・ダウンサンプリング・アイデア技術は他のＣＣＤセンサ解像度についてｎ／８（ｎ＝１，…，７）ダウンサンプリング比を提供することができることにも留意願いたい。

【００６６】アップサンプリング

画像のズーミング用の切り取られた画像を表示することもアップサンプリング方式を用いる。ダウンサンプリングと反対のアプローチはエレガント・ツールを提供する。最初の場合には、８×８ＤＣＴ係数は、Ｎ×Ｍ係数（Ｎ，Ｍ＞８）のブロックを形成するようにゼロで（実質的に）垂直および水平に拡大される。このブロックでは、サイズがＮ×ＭのＩＤＣＴが実行され、空間ドメインにＮ×Ｍサンプルを生成する。現在、ほとんどの画像パイプライン動作は標準化されていない。プログラム可能ＤＳＣエンジンを持つことは、ソフトウェアをグレードアップして新しい標準を構成したり画像パイプライン品質を向上する能力を提供する。未使用の性能は、ヒューマン・インターフェース，音声注釈，オーディオ記録／圧縮，モデム，ワイヤレス通信などのような他のタスク専用とすることもできる。図２７は、ＣＦＡ補間，ホワイトバランス，カラー補正，色調スケーリング，ガンマ補正，ＲＧＢからＹＣｒＣｂへの変換，エッジ強調，エッジ検出，カラー・ブーストおよびＪＰＥＧ圧縮の準備における誤りカラー抑制を含む前処理機能ブロック図を示す。以下のセクションは、ＣＦＡ補間に関する好ましい実施例を説明する。

【００６７】折返しを低減したＣＦＡ補間

バイエル・パターン（図７ａ）用の好ましい実施例のＣＦＡ補間は、緑チャンネルからの高周波を用いて赤および青チャンネル補間を修正して、他のカラー・チャンネルの信号を用いることによって画像内のエッジでの折返し成分を低減する。これによって、人工物が低減され、シャープさが向上し、更なる後処理が避けられる。実際、以下のような手順である。

（１）緑チャンネルに補間を適用する（いかなる補間方

法でも）。これは緑平面を生成する。
（２）緑チャンネルにおいてエッジを検出する（勾配または他の方法によって）。
（３）緑チャンネルのハイパス成分を計算する（任意のハイパスフィルタを備えたフィルタ）。
（４）赤チャンネルに補間を適用する（いかなる補間方法でも）。これは赤平面を生成する。
（５）（３）のハイパス成分を（重付け係数で）赤チャンネルに加算する。
（６）青チャンネルに補間を適用する（いかなる補間方法でも）。これは青平面を生成する。
（７）（３）のハイパス成分を（重付け係数で）青チャンネルに加算する。
【００６８】それで、最終画像は３つのカラー平面からなる。すなわち、ステップ（１）からの緑平面とステップ（５）からの赤平面とステップ（７）からの青平面とである。すなわち、最終画像の一画素について、緑の強度はステップ（３）からの緑平面の対応画素の値となるようにとられ、赤の強度はステップ（５）からの修正赤平面の対応画素の値となるようにとられ、青強度はステップ（７）からの修正青平面の対応画素の値となるようにとられる。前記の理論的な分析：各ＣＣＤ画素はその画素の空間的広がりにわたる入射光信号を平均化し、こうして、そのＣＣＤは入射光信号のローパスフィルタリングに画素サイズの逆数であるカットオフ周波数を効率的に提供する。また、画素についてのカラーフィルタによる画素アレイのサブサンプリングは、各カラー平面における折返しに導く。実際、赤と青については、サブサンプリングは各方向において２の係数によるものであり、それで、周波数スペクトラムは各方向において最大周波数の半分で折れる。こうして、赤および青のベースバンド・スペクトラム領域は原アレイ・スペクトラム領域の各４分の１である（赤および青サンプリングが原アレイの各４分の１であることを反映している）。緑については、スペクトラムの折れが斜め方向で赤および青と同じ大きさの$2^{1/2}$の距離であるという点で、サブサンプリングは半分だけしか良好ではない。緑のベースバンド・スペクトラムは、原アレイ・スペクトラムの２分の１の領域である。
【００６９】エッジでのカラー縁取りは、折返し問題である。また、たとえ折返しが存在していなくとも、非類似のベースバンド・スペクトラムもカラー縁取りに導く。実際、折返しが必ずしも一つのカラー・バンド画像では見えないが、３つのカラー成分を組み合わせて１つのカラー画像にすることに影響があることは明白となる。赤と緑と青との間のサンプリング格子のシフトは、折返し信号成分の位相シフトを引き起こす。一次元の例はこれを明らかにする。一次元離散信号ｆ（ｎ）と、それぞれ２の係数によるが偶数番目のサンプルの一つと奇数番目のサンプルの一つの（それで、１つのサンプルによってサンプリング格子のシフトがある）２つのサブサンプリングとを仮定する。
【００７０】
【数６】$f_{even}(2m)=f(2m)$
$f_{even}(2m+1)=0$
$f_{odd}(2m)=0$
$f_{odd}(2m+1)=f(2m+1)$
【００７１】もちろん、$f(n)=f_{even}(n)+f_{odd}(n)$である。$F(z)$を$f(n)$のｚ変換とし、$F_{even}(z)$を$f_{even}(n)$のｚ変換とし、$F_{odd}(z)$を$f_{odd}(n)$のｚ変換とする。ここで、$F_{even}(z)$はｚの偶数関数（ｚの偶数乗のみ）であり$F_{odd}(z)$はｚの奇数関数（ｚの奇数乗のみ）であることに留意願いたい。
【００７２】
【数７】$F_{even}(z)=\{F(z)+F(-z)\}/2$
$F_{odd}(z)=\{F(z)-F(-z)\}/2$
【００７３】$F(-z)$は、折返しに対応し、また、反対符号すなわちπの位相シフトで現れる。
【００７４】カラー縁取りは折返し成分のπの位相シフトによって低減することができる。しかしながら、これを達成するのは非常に困難である。というのは、たった一つの利用できる信号は原信号と折返し信号との和であるからである。したがって、好ましい実施例は別のアプローチを有する。２つ（または、それ以上）のサブサンプルされた信号（すなわち、赤，緑および青）が同一の特性（例えば、灰色スケール画像についてのような）を有している限り、原画像の完全再構築は、サブサンプルされた信号を加えさえすれば達成できる。しかしながら、ＣＦＡ補間においては、一般に、サブサンプルされた信号は異なるカラー・バンドから生じている。折返しのエラーは、異なるカラー・バンドの補間信号が誤って配列されたエッジで特に目に見えるものとなる。したがって、好ましい実施例は、サブサンプルされた信号の他のものを使うことによってエッジのみでの折返し成分を低減することによって、エッジでのカラー縁取りを阻止する。これによって、人工物が低減でき、シャープさを向上し、後処理が加わるのを避けるものとなる。
【００７５】とりわけ、バイエル・パターンＣＦＡについては、緑チャンネルは赤および青チャンネルよりも高いカットオフ周波数を有しており、それ故に、緑チャンネルの折返しはさほど重要なものではない。補償されるべき折返し信号はハイパス信号であり、それは緑チャンネルのハイパス成分として見積もられ、これは赤および青チャンネルに（緑のサブサンプリング格子に関する赤および青のサブサンプリング格子のオフセットによる位相シフトにより減算されるよりもむしろ）加算される。ハイパス緑成分は、赤および青のサブサンプリングに加算されるに先立って、スケール係数で乗算されることができる。信号は、赤，青を補間している間かその後に加算される。

【００７６】中間色相を適応するＣＦＡ補間

代わりのＣＦＡ補間の好ましい実施例は、まず、５×５ＦＩＲフィルタを使ってバイエル・パターン緑を補間し、その後、補間された緑を用いてそれぞれ２つのステップで赤および青を補間する。すなわち、まず、斜めに補間して原緑パターンに類似するパターンを形成し（この補間は緑による正規化を用いて高周波数を見積り）、その後、４つの最も近い隣接補間を適用して（緑の正規化を再度用いて高周波数を見積もって）赤または青平面を完了する。以下のようにｙがアレイの行番号でｘが列番号である画素位置（ｙ，ｘ）についてのＣＦＡ値をより明白に表わす。すなわち、ｙおよびｘが共に偶数である画素位置（ｙ，ｘ）での赤値Ｒ（ｙ，ｘ）と、ｙおよびｘが共に奇数である青値Ｂ（ｙ，ｘ）と、他の位置すなわちｙ＋ｘが奇数である緑値ｇ（ｙ，ｘ）である。

【００７７】まず、Ｇ＾（ｙ，ｘ）が緑平面補間の結果生じる画素位置（ｙ，ｘ）での緑値を表わすものとし、これはすべての画素位置（ｙ，ｘ）について定義される。この補間は、以下のセクションのエッジ保存補間を含む様々な方法によって行うことができる。多くの補間は原緑値を変えない、すなわち、Ｇが元々定義されると（すなわち、ｙ＋ｘが奇数であると）、Ｇ＾（ｙ，ｘ）＝Ｇ（ｙ，ｘ）が真であるということに留意願いたい。次に、青についてラベル付けされるとともに補間寄与を示すのに矢印を用いる図２８に例示されるように、２つのステップで赤補間および青補間をそれぞれ定義する。まず、赤ステップである。Ｒ（ｙ，ｘ）は、すでに、ｍおよびｎを整数としてｙ＝２ｍおよびｘ＝２ｎで画素位置（ｙ，ｘ）について定義されており、それで、第１に、ｙ＝２ｍ＋１およびｘ＝２ｎ＋１についてＲ＾（ｙ，ｘ）を定義する。

【００７８】

【数８】R^(y, x)=G^(y, x){R(y-1, x-1)/G^(y-1, x-1)+R(y-1, x+1)/G^(y-1, x+1)+R(y+1, x-1)/G^(y+1, x-1)+R(y+1, x+1)/G^(y+1, x+1)}/4

【００７９】これは、赤平面をＢ（ｙ，ｘ）が定義された画素に補間する。（図２８は、青についての類似する補間を例示する。）この補間は本質的には対応緑値によって各位置で正規化された値で（ｙ，ｘ）周りの３×３正方形の４つの角で赤値を平均化する。緑値に閾値未満のものがあれば、正規化を省略して赤値を平均することだけ行われる。同じ緑値が用いられているので、最初の赤ステップと並列して最初の青ステップを行う。最初の青ステップ：Ｂ（ｙ，ｘ）は、ｍおよびｎを整数としてｙ＝２ｍ＋１およびｘ＝２ｎ＋１で画素位置（ｙ，ｘ）についてすでに定義されているので、まず、ｙ＝２ｍおよびｘ＝２ｎについてＢ＾（ｙ，ｘ）を定義する。

【００８０】

【数９】B^(y, x)=G^(y, x){B(y-1, x-1)/G^(y-1, x-1)+B(y-1, x+1)/G^(y-1, x+1)+B(y+1, x-1)/G^(y+1, x-1)+B(y+1, x+1)/G^(y+1, x+1)}/4

【００８１】これは、図２８の左手部分に例示されるようにＲ（ｙ，ｘ）が定義された画素に青平面を補間する。ここでも、この補間は、本質的に、対応緑値によって各位置で正規化された値で（ｙ，ｘ）周りの３×３正方形の４つの角で青値を平均化する。第２の赤ステップ。ｙ＋ｘが奇数である（ｙ＝２ｍおよびｘ＝２ｎ＋１かｙ＝２ｍ＋１およびｘ＝２ｎである）場合のＲ＾（ｙ，ｘ）を定義する。

【００８２】

【数１０】R^(y, x)=G^(y, x)[R^(y-1, x)/G^(y-1, x)+R^(y, x+1)/G^(y, x+1)+R^(y+1, x)/G^(y+1, x)+R^(y, x+1)/G^(y, x+1)]/4

【００８３】この第２のステップは、第１のステップによって定義された赤平面部分をＧ（ｙ，ｘ）が定義される画素に補間する。ここでもまた、この補間は、本質的に、対応緑値によって各位置で正規化された値で（ｙ，ｘ）の４つの隣接画素での赤値を平均化する。第２の青ステップ。ｙ＋ｘが奇数（ｙ＝２ｍおよびｘ＝２ｎ＋１かｙ＝２ｍ＋１およびｘ＝２ｎ）であるものについて定義する。

【００８４】

【数１１】B^(y, x)=G^(y, x){B^(y-1, x)/G^(y-1, x)+B^(y, x+1)/G^(y, x+1)+B^(y+1, x)/G^(y+1, x)+B^(y, x+1)/G^(y, x+1)}/4

【００８５】この第２のステップは、第１のステップによって定義された青平面部分をＧ（ｙ，ｘ）が定義される画素に補間する。ここでもまた、この補間は、本質的には、対応緑値によって各位置で正規化される値で（ｙ，ｘ）の４つの隣接画素で青値を平均化する。最終カラー画像は、３つの補間されたカラー平面Ｇ＾（ｙ，ｘ），Ｒ＾（ｙ，ｘ），Ｂ＾（ｙ，ｘ）によって定義される。Ｇ＾（ｙ，ｘ）について用いられる特定の補間は、Ｒ＾（ｙ，ｘ）およびＢ＾（ｙ，ｘ）のために用いられる２ステップ補間用の正規化において反映される。

【００８６】エッジを保存するＣＦＡ補間

代わりのＣＦＡ補間の好ましい実施例は、補間された画素緑値を最も近い隣接画素緑値と比較し、また、補間された値が範囲外にあれば補間された値を隣接値と置き換えることによって、（小さな）ＦＩＲフィルタと保存エッジとでバイエル・パターン緑を補間する。図２９は緑補間を例示する。この緑補間の後に、赤平面および青平面を補間する。とりわけ、第１に、各画素（ｙ，ｘ）で以下の５×５ＦＩＲフィルタをｘ＋ｙが奇数である画素（ｙ，ｘ）上で定義されるＧ（ｙ，ｘ）に適用し、すべての（ｙ，ｘ）について定義されるＧ１（ｙ，ｘ）を生成する。

【００８７】

【数１２】

$$1/200\begin{vmatrix} 0 & -11 & 0 & -11 & 0 \\ -11 & 0 & 72 & 0 & -11 \\ 0 & 72 & 200 & 72 & 0 \\ -11 & 0 & 72 & 0 & -11 \\ 0 & -11 & 0 & -11 & 0 \end{vmatrix}$$

【００８８】２００の中央エントリは、Ｇ(ｙ，ｘ)がＣＦＡにおいて定義される（ｙ，ｘ）についてＧ１(ｙ，ｘ)＝Ｇ(ｙ，ｘ)を意味するに過ぎない。緑値は０～２５５の範囲にあることと負値は０に切り捨てられることとに留意願いたい。もちろん、他のＦＩＲフィルタを用いることもできるが、このものが簡単で有効である。次に、Ｇ１(ｙ，ｘ)が補間される（ｙ，ｘ）について、４つの最も近い隣接値Ｇ(ｙ±１，ｘ)，Ｇ(ｙ，ｘ±１)を考慮して、最大および最小の値を破棄する。ＡおよびＢを残りの２つの最も近い隣接値であって、ＢがＡ以下であるとする。そのとき、最終補間緑値Ｇ＾(ｙ，ｘ)は以下のように定義される。

【数１３】

$$G\hat{}(y,x)=\begin{cases} A & G1(y,x)<A\text{のとき} \\ G1(y,x) & A\leqq G1(y,x)\leqq B\text{のとき} \\ B & B<G1(y,x)\text{のとき} \end{cases}$$

【００８９】これは、補間された値を隣接画素値の中間範囲に固定し、一つのエッジを超えた最も近い隣接画素が補間画素値を薄めないものとする。図２９は緑補間全体を示す。赤補間および青補間によって画像を完成させる。赤補間および青補間はそれぞれ、単一ステップの補間であってもよいし、エッジ保存された緑値を用いる前述のセクションにおいて説明したような２ステップ補間であってもよいし、他の種類の補間であってもよい。

【００９０】ＣＦＡ補間とノイズフィルタリング

好ましい実施例は、一体化されたアプローチでノイズを制限するローパスフィルタリングが続いて行われるＣＦＡ補間に必要なラインメモリに保存する。とりわけ、ＣＦＡ補間は、典型的には、図３０に例示されるように、間にラインメモリを有する水平補間ブロックおよび垂直補間ブロックを含む。水平補間ブロックは、ＣＦＡ信号の行の入力と、２つのトグルスイッチと、２つのゼロ挿入サブブロックと、２つの３タップＦＩＲフィルタ（係数０．５，１．０，０．５）と、２つの出力（各カラーにつき１つの出力）とを有する。ＦＩＲフィルタの各々は、入力カラー値を再生するに過ぎず、また、挿入されたゼロの代わりに連続する入力カラー値の平均を入れる。２つのサブブロックのゼロ挿入およびトグル・タイミングは、互いに交互になる。水平補間ブロックのブロック図が図３１に示され、生データＲ／Ｇ／Ｒ／Ｇ／Ｒ．．．を有し、このブロックにおいて、行補間された赤および緑信号が出力される。生データ入力の行がＢ／Ｇ／Ｂ／Ｇ／Ｂ．．．であるならば、補間された青および緑信号が出力される。ライン（行）メモリは、データを垂直補間ブロックで補間するために、１ＣＦＡライン（行）周期だけデータを遅延させる。図３２は、４つのラインメモリとそのメモリの入出力データとを示す。行データが入力するにつれて増大する（偶数）行番号を示すｍと列番号を示すｎについてのＲ／Ｇ／Ｒ／Ｇ／．．．生データの入力行の場合には、入力および出力データは、

Input__A=R(m, n)

Output__A1=Input__A=R(m, n)

Output__A2=G(m-1, n)　これは生データＧ／Ｂ／Ｇ／Ｂ．．．行の前の行からの補間された緑であった。

Output__A3=R(m-2, n)　これは生データＲ／Ｇ／Ｒ／Ｇ／．．．行の２つ前の行からの補間された赤であった。

Input__B=G(m, n)

Output__B1=Input__B=G(m, n)

Output__B2=B(m-1, n)　これは生データＧ／Ｂ／Ｇ／Ｂ．．．行の前の行からの補間された青であった。

Output__B3=G(m-2, n)　これは生データＲ／Ｇ／Ｒ／Ｇ／．．．行の２つ前の行からの補間された緑であった。

これによって、垂直補間のために赤の２行Ｒ(ｍ，ｎ)，Ｒ(ｍ－２，ｎ)が備わり、赤のｍ－１行を生成し、また、垂直補間を必要としない緑行Ｇ(ｍ，ｎ)，Ｇ(ｍ－１，ｎ)，Ｇ(ｍ－２，ｎ)も備わるものとなる。

【００９１】Ｇ／Ｂ／Ｇ／Ｂ／．．．生データの次の入力行（行ｍ＋１）によって以下の入力および出力データが導かれる。

Input__A=G(m+1, n)

Output__A1=Input__A=G(m+1, n)

Output__A2=R(m, n)　これは生データＲ／Ｇ／Ｒ／Ｇ／．．．行の前の行から補間された赤であった。

Output__A3=G(m-1, n)　これは生データＧ／Ｂ／Ｇ／Ｂ／．．．行の２つ前の行から補間された緑であった。

Input__B=B(m+1, n)

Output__B1=Input__B=B(m+1, n)

Output__B2=G(m, n)　これは生データＲ／Ｇ／Ｒ／Ｇ／．．．行の前の行から補間された緑であった。

Output__B3=B(m-1, n)　これは生データＧ／Ｂ／Ｇ／Ｂ／．．．行の２つ前の行から補間された青であった。

これによって、垂直補間のために２つの青の行Ｂ(ｍ＋１，ｎ)，Ｂ(ｍ－１，ｎ)が備わり、ｍ行の青を定義し、また、垂直補間を必要としない緑行Ｇ(ｍ＋１，ｎ)，Ｇ(ｍ，ｎ)，Ｇ(ｍ－１，ｎ)も備わる。

【００９２】図３３は垂直補間用の組合せを示す。とりわけ、行ｍ出力（行ｍ＋１入力）については組合せは（図３３ｂ）、

緑はＧ(ｍ，ｎ)

赤はＲ(ｍ，ｎ)

青は（Ｂ(ｍ－１，ｎ)＋Ｂ(ｍ＋１，ｎ)）／２

である。また、行ｍ－１出力（行ｍ入力）については組

合せは（図３３ａ）、
緑はG(m－１，n)
赤は（R(m，n)＋R(m－２，n)）／２
青はB(m－１，n)
である。

【００９３】図３３が例示するように、垂直ローパスノイズフィルタは、３つの緑出力（行m入力に対してG(m－２，n)，G(m－１，n)およびG(m，n)と、行m＋１入力に対してG(m－１，n)，G(m，n)およびG(m＋１，n)）に直接適用することができるが、図３２の４つのラインメモリが十分なライン（行）を出力しないので、赤と青とは垂直にフィルタすることができない。むしろ、図３４に例示されるように、８つのラインメモリが必要である。

【００９４】図３５ａおよび図３５ｂは、緑垂直ノイズ低減フィルタ・ブロックＡと緑ノイズ・ブロックＢと青／赤緑ノイズ差分ブロックＣと赤／青緑ノイズ総和ブロックＤとを含む垂直補間およびローパスノイズフィルタリングの好ましい実施例の組合せを例示する。図３５ａおよび図３５ｂの好ましい実施例に対する６つの入力は、図３０から図３２の水平補間および４つのラインメモリの出力であり、このように、図３４の公知の垂直補間フィルタへの入力と同じである。この補間とノイズフィルタリングとをプログラム可能プロセッサ上で実施するためには、図３４の８つのラインメモリが図３０から図３２の４つのラインメモリの２倍のプロセッサ・メモリ空間をとり、これは大きなメモリ空間となり得る。２メガピクセル（１９２０×１０８０画素）ＣＣＤのような大きなＣＦＡについては、ラインメモリは１～２キロバイトであり、差はプロセッサ・メモリの４～８キロバイトとなる。

【００９５】より詳細には、図３５ａは、図３２の水平補間＋４つのラインメモリへのmが偶数である入力列m（生ＣＦＡデータR／G／R／G／．．．）の場合のノイズ低減および垂直補間を例示する。図３５ａの左手エッジでの６つの（水平に補間された）入力は、R(m，n)，G(m－１，n)，R(m－２，n)，G(m，n)，B(m－１，n)およびG(m－２，n)であり（すなわち、図３２における出力）、また、出力は、行m－１についてノイズ低減されたカラー、すなわち、R”(m－１，n)，G”(m－１，n)およびB”(m－１，n)である。まず、垂直補間（図３５ａの左手部分）は、R(m，n)およびR(m－２，n)を平均化してR(m－１，n)を生成し、G(m－１，n)およびB(m－１，n)は入力としてすでに存在する。そして、ノイズ低減フィルタ（図３５ａの右手部分におけるブロックＡ）は、垂直にローパスフィルタリングされた緑G”(m－１，n)を生成して出力する。

【００９６】

【数１４】G”(m-1，n)＝[G(m，n)＋２＊G(m-1，n)＋G(m-2，n)]/4

【００９７】次に、ブロックＢは、Delta__GをGとG”との差として生成する、すなわち、Delta__GはGの垂直高周波部分である。

【００９８】

【数１５】

Delta__G(m-1，n)=G(m-1，n)－G”(m-1，n)

【００９９】Gは、バイエルＣＦＡにおけるＢおよびＲの２倍の頻度でサンプルされるので、Gの直接高周波見積りはＢおよびＲのそれよりも良いように思え、このように、好ましい実施例はノイズ低減用の減算にDelta__Gを用いる。垂直平均［G(m＋１，n)－G(m－１，n)］／２とG”(m，n)との差が－Delta__G(m，n)と等しくなり、それにより、垂直に補間され（平均化され）かつローパスフィルタリングされるべきＲおよびＢについて、ＲおよびＢから減算されるべきGによって提供される高周波数見積りが反対の符号を有するものとなることに留意願いたい。

【０１００】このように、ブロックＣは、Delta__GをＢから減算して行m－１についてB”を生成する。というのも、Ｂはｎ－１について垂直に補間されないからである。

【０１０１】

【数１６】

B”(m-1，n)＝B(m-1，n)－Delta__G(m-1，n)

【０１０２】本質的に、Gの垂直高周波部はＢの垂直高周波部についての見積りとして用いられ、Ｂの直接垂直ローパスフィルタリングは適用されない。

【０１０３】続いて、ブロックＤは、Delta__GをＲに加算して行m－１についてR”を生成する。というのも、Ｒは垂直に補間されなかったからである。

【０１０４】

【数１７】

R”(m-1，n)＝R(m-1，n)＋Delta__G(m-1，n)

【０１０５】ここでも、Gの垂直高周波部がＲの高周波部の代わりに用いられ、垂直平均化がR(m－１，n)を生成するので、Delta__Gの反対符号が高周波数見積りを減算するのに用いられる。

【０１０６】このように、ノイズが低減されフィルタリングされた３つのカラー出力行m－１は、先のG”(m－１，n)，R”(m－１，n)およびB”(m－１，n)である。同様に、入力行m＋１からの出力行m（ここでもmは偶数）および生ＣＦＡデータG／B／G／B／．．．について、６つの（水平に補間された）入力は、G(m＋１，n)，R(m，n)，G(m－１，n)，B(m＋１，n)，G(m，n)およびB(m－１，n)であり、出力は、行mについてノイズの低減されたカラーR”(m，n)，G”(m，n)およびB”(m，n)である。垂直補間（図３５ｂの左手部分）は、B(m＋１，n)とB(m－１，n)とを平均化してB(m，n)を生成

し、G(m, n)とR(m, n)とはすでに入力として存在する。ノイズ低減フィルタ（図３５ｂの右手部分）ブロックＡは、再度垂直にローパスフィルタリングされた緑G”(m, n)を次の通り生成する。

【０１０７】

【数１８】G”(m, n)＝{G(m+1, n)＋２＊G(m, n)＋G(m-1, n)}/4

【０１０８】次に、ブロックＢは、再度、Delta__Gと呼ばれるGの垂直高周波部分をGとG”との差分として生成する。

【０１０９】

【数１９】Delta__G(m, n)=G(m, n)－G”(m, n)

【０１１０】その後、ブロックＣは、再度、Delta__Gを（行ｍ－１出力についてはＢよりもむしろ）Ｒのみから減算してR”を生成する。

【０１１１】

【数２０】R”(m, n)＝R(m, n)－Delta__G(m, n)

【０１１２】このように、Gの高周波部分は、再度、Rのノイズのある部分についての見積りとして用いられ、また、Rの直接のノイズフィルタリングは適用されず、行mについてはDelta__Gが行ｍ－１に対して加算されるよりはむしろ減算される。実際、奇数行はRを垂直平均として定義しているので、Rについて、偶数行はDelta__Gを減算させ、奇数行はDelta__Gを加算させる。最後に、ブロックＤは、Delta__GをＢに加算してB”を生成する。

【０１１３】

【数２１】B”(m, n)＝B(m, n)＋Delta__G(m, n)

【０１１４】このように、Ｂの直接垂直ローパスフィルタリングの代わりに、Rについて、Delta__G垂直高周波数見積りは、行ごとに交代でＢに加算されまたＢから減算される。所定の行について、ＲおよびＢの一方は先行する行および後続する行の平均であるので、ＲおよびＢについてのDelta__G項は反対の符号を有することに留意願いたい。

【０１１５】要するに、好ましい実施例は、Gに基づく高周波数見積りを用いることによってたった４つのラインメモリでＣＦＡ水平補間，垂直補間およびローパスフィルタリングを模倣することができる。図３６ａから図３６ｂおよび図３７ａから図３７ｂは、Gの垂直ローパスフィルタリングが、図３５ａから図３５ｂの好ましい実施例の１／４，１／２，１／４の重付けとは異なる代替の実施例を例示している。

【０１１６】補色ＣＣＤ用のＣＦＡ補間

補色パターンＣＦＡ（図７ｂに例示される）のための好ましい実施例のＣＦＡ補間は、単純補間を、それに続くカラー不均衡の検出および調整による画像品質強調と組み合わせる。とりわけ、各画素で定義された最初の補間をすべての４つの補色値とし、色値をＹｅ（黄），Ｃｙ（シアン），Ｍｇ（マゼンタ）およびG（緑）と記す。

最初に、各画素で不均衡ファクターμを計算する。

【０１１７】

【数２２】μ＝Ｙｅ＋Ｃｙ－２＊Ｇ－Ｍｇ

【０１１８】この不均衡ファクターは、理想の画素カラー値と実際の画素カラー値との差を現す。実際に、赤値（Ｒ），緑値（Ｇ）および青値（Ｂ）に関しての補色値の定義は、Ｙｅ＝Ｒ＋Ｇ，Ｃｙ＝Ｇ＋ＢおよびＭｇ＝Ｂ＋Ｇである。それ故に、画素のカラー値について以下の関係が常に成立する。

【０１１９】

【数２３】Ｙｅ＋Ｃｙ＝２＊Ｇ＋Ｍｇ

【０１２０】このように、不均衡ファクターμは理想的には消失する。エッジが画素に近いとき、不均衡はＣＦＡにおける４つのカラー・サンプルの各々の空間的差分により生じ得る。好ましい実施例は、不均衡を検出し、各カラー値を変更することによって調整する。

【０１２１】

【数２４】Ｙｅ’＝Ｙｅ－μ／４

Ｃｙ’＝Ｃｙ－μ／４

Ｍｇ’＝Ｍｇ＋μ／４

Ｇ’＝Ｇ＋μ／８

【０１２２】その後、これらの変更された補色は最終画像を形成するのに用いられる。図３８は、不均衡ファクターを用いた強調（エンハンスメント）についての全体的なフローを明示する。もちろん、Ｙｅ’＋Ｃｙ’＝２＊Ｇ’＋Ｍｇ’であるならば、－１／４，－１／４，１／４および１／８以外のスケール因数を不均衡ファクターに適用することもできる。

【０１２３】ホワイトバランス

「ホワイトバランシング」という語は、典型的には、アルゴリズムを記述するのに用いられ、それは、カメラが現在作動している光源に関してカメラのホワイトポイントを補正する。真の光スペクトラムの見積りが非常に困難であるので、たいていのアプローチの目的は、灰色の対象物についてはすべてのカラー・チャンネルについての画素強度がほとんど同じものとなるように、（ＲＧＢカラーフィルタに基づくＣＣＤであるとして）赤および青チャンネルの出力を補正することである。最も普通の技術は、基本的には、平均エネルギーまたは単純に各チャンネルの中間値を計算する。平均の計算は、赤についてはN個のローカル・ウインドウW_j（ｊ＝１，２，．．．，Ｎ）において実行される。

【０１２４】

【数２５】$R_j=\Sigma_{k\in w_j} r(k)$

【０１２５】ここで、ｒ(ｋ)は赤チャンネル用のデジタル信号を示す。青および緑カラー・チャンネルについては、同様に、平均B_jおよび平均G_jが計算される。チャンネル間の不均衡は、緑対赤の比および緑対青の比によって与えられる

【０１２６】

【数２６】$WBR = \Sigma_j G_j / \Sigma_j R_j$

$WBB = \Sigma_j G_j / \Sigma_j B_j$

【０１２７】が、赤および青チャンネル用の補正乗数としてそれぞれ用いられる。

【０１２８】

【数２７】$r'(k) = WBR\, r(k)$

$b'(k) = WBB\, b(k)$

【０１２９】このアプローチには多くの異なる種類が存在し、それらはすべて、強度から独立した乗算因数ＷＢＲ，ＷＢＢを計算する。

【０１３０】このアプローチは、いくつかの仮定が有効でありさえすれば、成り立つ。第１に、センサ応答は入力強度範囲全体にわたって良好に配列されると仮定する。換言すれば、緑応答曲線は、因数を乗算された赤（青）応答曲線に等しい。センサ（ＣＣＤ）特性を見ることは、この仮定が成立するのではないということを指示する。高い光強度については、センサは飽和し、一方、非常に低い光強度では、センサ反応（特に、青チャンネル）は非常に小さい。また、センサの非線形性とセンサ反応に関するカラー・チャンネルの不均衡と光源とは、同時に取り扱われる。結果としての人工物は、非常に明るい領域でのマゼンタ・カラーを含み、そこでは、「カラー」が白くなったり暗い領域で違うカラーになったりする。センサ出力での、例えば赤カラー・チャンネルについての画素強度は、以下のようにモデル化することができる。

【０１３１】

【数２８】$r(k) = \int I(\lambda)\beta(k, \lambda) f_R(\lambda)\alpha(I, \lambda)\, d\lambda$

【０１３２】ここで、λは波長を示し、Ｉ（λ）は光源のスペクトラムを示し、$\beta(x, \lambda)$は観察下にある対象物の反射率を示し、$f_R(\lambda)$はＣＣＤ画素をカバーする赤カラーフィルタのスペクトラムの感度を示し、$\alpha(I, \lambda)$は光子を電子に変換するＣＣＤの強度および波長に依存する効率を示す。

【０１３３】典型的なＣＣＤセンサのカラーフィルタ$f_R(\lambda)$（と$f_G(\lambda)$および$f_B(\lambda)$）のスペクトラム応答曲線に関してのみ、出力信号は異なる。

【０１３４】

【数２９】

$WBR = \int f_G(\lambda)\, d\lambda / \int f_R(\lambda)\, d\lambda = 1.09$

$WBB = \int f_G(\lambda)\, d\lambda / \int f_B(\lambda)\, d\lambda = 1.34$

【０１３５】典型的なＣＣＤの応答を用いて、完全白色光源（スペクトラムＩ（λ）がフラットである）と完全に白い対象物（反射光のスペクトラムが照射光のスペクトラムと同じであって、$\beta(k, \lambda) = 1$を意味する）とを想定し、また、$\alpha(I, \lambda)$を無視（波長に依存する量子効果がない）して、それらの値が得られる。特に、青チャンネルは、同じ強度で緑または赤よりも小さな応答を示す。センサの非線形量子効果は別の影響である。入力

強度にわたっての典型的なＳ字型のセンサ反応が図３９ａに示されている。また、各チャンネルにおけるセンサ反応は光源のスペクトラムに依存する。

【０１３６】こうして、好ましい実施例のホワイトバランシングは非整列および非線形性を考慮する。典型的な光源は、可視スペクトラムにわたってフラットではなく、あるスペクトラムの帯域においてより高いエネルギーを有する傾向にある。この効果は、観察されるセンサ反応に影響し、理想的には、それがホワイトポイント補償によって補正されるが、それは補正行列に基づく。チャンネルの独立したバランシングは、この効果を先に概要を述べたように取り扱うことができない。数学的な記述を簡単にするために、図３９ａのＳ字型応答曲線を区分線形部分で近似する。３つの部分は光の状態を３つのカテゴリ（非常に低い強度と通常の強度と非常に明るい光）に分ける。図３９ｂは単一の乗数を適用する効果を示す。緑信号に関して、青信号の増幅は、低い光の状態においてはあまりに小さすぎ、一方、非常に明るい状態においてはその乗数は大きすぎる。ファクターを低減すると、成分間でのオフセットが残り、違う色として目に見える。したがって、３つのすべての応答曲線を整列するための補正条件は、異なって見え、センサ特性を反映しなければならない。

【０１３７】好ましい実施例のホワイトバランシングは、２つの別々の方式に分けられる。一つはイメージャに依存する調整を考慮し、他方は光源に関係する。一般的なものとして制限されるわけではないが、Ｓ字型応答曲線は、以下の３つの区分線形部分で近似される。より多くの部分とすれば、制度が向上するが、基本的な概念は変わらない。第１の領域（非常に低い強度）と青チャンネルについて、ｓを応答としｘを入力強度としてモデルが読み取られる。

【０１３８】

【数３０】$s_{B,1} = a_{B,1} x$

【０１３９】第２の領域をモデル化するには、乗数およびオフセットが必要である。

【０１４０】

【数３１】$s_{B,2} = a_{B,2} x + b_{B,2}$

【０１４１】オフセット項は、応答曲線が領域１から領域２への移行ポイントx_1で連続する必要があるという制約によって決定される。

【０１４２】

【数３２】$s_{B,1}(x_1) = s_{B,2}(x_1)$

したがって、$b_{B,2} = (a_{B,1} - a_{B,2})\, x_1$

【０１４３】領域３の線形モデルについてのパラメータ

【０１４４】

【数３３】$s_{B,3} = a_{B,3} x + b_{B,3}$

【０１４５】が完全に決定される。というのも、最大出力は最大入力x_{max}と同じでなければならず、また、応答曲線は接続ポイントx_2で連続する必要があるからで

ある。

【０１４６】

【数３４】 $x_{mx}=a_{B,3}x_{mx}+b_{B,3}$

$s_{B,2}(x_2)=s_{B,3}(x_2)$

$a_{B,3}=(s_{B,2}(x_2)-x_{mx})/(x_2-x_{mx})$

$b_{B,3}=(1-a_{B,3})\ x_{mx}$

【０１４７】こうして、各カラー成分について応答曲線の近似を特定するパラメータはa_1，a_2，x_1およびx_2である。x_{mx}は、入力信号のビット解像度によって特定されるので、自由パラメータではない。

【０１４８】好ましい実施例のホワイトバランシングは、各領域について異なる乗数を適用する。１つの領域から次の領域への連続する移行については、更なるオフセットが要求される。領域の数は任意であるが、一般性を失うことなく、以下の等式では３つだけの領域が考慮される。領域１についての緑に関する青の補正項は、

【０１４９】

【数３５】

$$WBB_1 = a_{G,1}/a_{B,1} \approx G_1/B_1$$

【０１５０】でなければならない。ここで、（G_1およびB_1用の）ウインドウ１は、領域１の強度をもつ画素を有する。このように、領域１にある入力強度値は、補正された出力

【０１５１】

【数３６】 $b'(k)=WBB_1 b(k)$

【０１５２】を得る。領域２についてのバランシング乗数に基づいて、

【０１５３】

【数３７】

$$WBB_2 = a_{G,2}/a_{B,2} \approx G_2/B_2$$

【０１５４】ホワイト・バランシングは、領域２における値の更なるオフセットを考慮しなければならず、

【０１５５】

【数３８】 $b'(k)=WBB_2 b(k)+WBOB_2$

ここで、

$WBOB_2=(WBB_1-WBB_2)\ x_1$

【０１５６】第３の領域について、WBB_3を明瞭に特定することができないことを除いて、計算は基本的に同じであるが、増幅は最大値x_{mx}によって決定される。

【０１５７】

【数３９】 $b'(k)=WBB_3 b(k)+WBOB_3$

ここで、

$WBB_3=(x_{mx}-(WBB_2 x_2+WBOB_2))/(x_{mx}-x_2)$

$WBOB_3=(1-a_{B,3})\ x_{mx}$

【０１５８】実施のために、システムは、Ｎ－１の領域について適当なホワイトバランシング乗数WBB_1を決定しなければならない。これらの値に基づいて、残りのオフセット値WBOBおよび最後の領域用の乗数が計算される。移行ポイントの位置は演繹的に特定される。ホワイトバランシング自体は、入力画素の強度値に基づいて領域を選択し、適当な利得およびオフセットをその値に適用する。

【０１５９】

【数４０】

$$b'(k)=\begin{cases} WBB_1 * b(k) & b(k)\leq x_1 \\ WBB_2 * b(k)+WBOB_2 & x_1 < b(k)\leq x_2 \\ WBB_3 * b(k)+WBOB_3 & x_2 < b(k) \end{cases}$$

【０１６０】赤チャンネルについて同様の乗数も加える。

【０１６１】ＣＣＤ出力信号の全ダイナミックレンジは、ＣＣＤにおいて捕獲される光子の数に影響するので、開口およびシャッターから独立である。しかしながら、アナログ利得または処理に先立つデジタル利得は、信号をシフトさせ、避けられるべきである。利得（デジタル）αが適用される必要のある場合は、この利得はホワイトバランシング方法に含まれる。利得は、最大入力値x_{mx}を出力値$\alpha * x_{mx}$にマッピングする。スケーリングされた応答曲線は、スケーリングされていないものと同じに振る舞い、スケーリングされた信号が$\alpha * x_{mx}$で飽和することを意味している。

【０１６２】

【数４１】 $WBB_1 := \alpha * WBB_1$

$WBB_2 := \alpha * WBB_2$

【０１６３】を代入のこと。このようにして、先のセクションにおける等式は、以下を除いて変わっていない。

【０１６４】

【数４２】 $WBOB_3=(\alpha - a_{B,3})\ x_{mx}$

【０１６５】線形化の後に、信号は光源を反映して調整を受けることができる。これはまた、ホワイトポイント調整としても知られている。ここで、入力信号は、異なる光源の下で捕らえられたかのように見えるように変換される。例えば、画像は、明るい日光でとられた（Ｄ６５）が、カラー特性は室内状態（Ｄ５０タングステン）の下でとられているかのようである。

【０１６６】

【数４３】

$[R,G,B]D_{65}^T = I_{D65}^T * \beta * [f_R, f_G, f_B]^T$

$[R,G,B]D_{50}^T = I_{D50}^T * \beta * [f_R, f_G, f_B]^T$

【０１６７】ここで、I_{Dxx}は光スペクトラムをサンプリングするベクトルを示し、βは対象物の反射率を記述する斜行列であり、f_R，f_Gおよびf_BはＣＣＤ光フィルタのスペクトラム応答を示す。これらの等式に基づいて、Ｄ６５からＤ５０の下での信号に関して３×３変換

行列を計算することができる。

【０１６８】

【数４４】［R，G，B］D_{60}^{T}＝I_{D50}^{T}＊I_{D65}^{-1}＊［R，G，B］D_{65}^{T}

【０１６９】３×３変換行列

【０１７０】

【数４５】M_D＝I_{D50}^{T}＊I_{D65}^{-T}

【０１７１】は、オフラインで計算することができる。現実のシステムでは、異なる応答領域について平均を求めることはほとんど不可能である。したがって、簡単な解決策は、先の一体化の比において全体の値を計算し、それらを所定のセンサ測定に基づいて固定値で変更することである。

【０１７２】

【数４６】$WBB_1=\alpha_1$＊WBB

$WBB_2=\alpha_2$＊WBB

【０１７３】WBRについても同様である。移行ポイントもまた、前もって固定することができる。移行ポイントx_2いう１つだけの例外がある。稀な状況では、WBR値は非常に大きくて移行ポイントx_2で最大出力値を超える。その状況では、WBRが低減される必要があるか移行ポイントが低減される。図４０の図は、この技術の有効性の例を示す。赤構成要素は、緑構成要素に関して調整される。単一の乗数を用いることは、明るい領域において緑信号を超え、低い光領域において有効ではなく、一方、区分化されたホワイトバランシングがあらゆる強度の緑曲線にマッチする。

【０１７４】好ましい実施例のサイズ調整

あるサイズ（例えば、３２０×２４０画素）で捕獲された画像は、様々な蓄積または入出力フォーマットにマッチするために頻繁に別のサイズ（例えば、約２８８×２１６）に変換されなければならない。一般に、これには有理の因数N／Mによる分数のアップサンプリングまたはダウンサンプリングが必要とされ、例えば、３２０×２４０から２８８×２１６へのサイズ調整は９／１０のサイズ調整である。理論的には、サイズ調整は、Nによる何重もの補間と、反折返しフィルタとMによるデシメーションとに相当する。特に、サイズ調整は、M位相のKタップフィルタリングとM入力当りN出力の選定とで達成される。例えば、最上の水平線が画素入力を表し、また、水平長さ－３つの括弧が、指示された３つの入力に適用されて指示された出力を生成する３タップ・フィルタ核を表わす図４１ａに図示されるような３タップ・フィルタを用いる６３／６４の比でのサイズ調整を前置きとして考える。実際、フィルタ核は、長さ３－１／６３の支持で連続する関数ｆ（ｔ）であり、最大３つの入力を含むことができるものとする。図４１ｂを参照のこと。図４１ａにおいて連続する括弧の右にわずかにシフトしていることに留意願いたい。これは、６４入力から６３出力へのサイズ調整を表わす。というのも、フィルタ核の中心（それ故に、概数で表されていない出力位置）が、６３の出力が６４の入力にマッチするために、各出力について１+１／６３（＝６４／６３）画素位置だけ増大しなければならないからである。出力［０］（図４１ａにおいて最も左側の括弧によって表される）が入力の位置で真中に置かれ、また、それでｏｕｔｐ＿ｐｏｓ［ｊ］と記される概数で表されていない出力位置ｊが１+ｊ＊６４／６３に等しくなる。

【０１７５】フィルタ核は、時間０で中央に置かれる対称な連続関数ｆ（ｔ）として表される。例えば出力［０］は３つの核値ｆ（－１），ｆ（０），ｆ（１）を必要とする。各出力ポイントは，３つの核係数値の３つの入力画素値との内積（inner product）として計算される。出力［ｊ］用の中央入力ポイントはｒｏｕｎｄ（ｏｕｔｐｐｏｓ［ｊ］）で位置付けられ、ここで、ｒｏｕｎｄ()は四捨五入機能である。他の２つの入力ポイントはこの中央のポイントからの±１だけのオフセットである。中央フィルタ核係数値は、ｆ（ｒｏｕｎｄ（ｏｕｔｐ＿ｐｏｓ［ｊ］）－ｏｕｔｐ＿ｐｏｓ［ｊ］）であり、他のものは、この中央値ポイントの±１オフセットでのｆ()である。こうして、以下の表が、各出力に必要なの出力位置，係数核値および入力ポイントを示す。

【０１７６】

【表１】

出力	outp_pos	中央係数位置	入力ポイント
0	1	0	0,1,2
1	2 1/63	-1/63	1,2,3
2	3 2/63	-2/63	2,3,4
...	...	...	...
31	32 31/63	-31/63	31,32,33
32	33 32/63	31/63	33,34,35
33	34 33/63	30/63	34,35,36
...	...	...	...
61	62 61/63	2/63	62,63,64
62	63 62/63	1/63	63,64,65
63	65	0	64,65,66
...	...	...	...

【０１７７】この表は、各出力に関する望ましい係数位置および入力を示す。ｊ＝６３の場合は、核の中央が入力と整列するということでｊ＝０の場合と類似するが、出力位置および入力インデックスは６４だけシフトしているということに留意願いたい。ｊ＝３２では入力パターンに変化があることに注意し、ｊが３１以下では、出力[ｊ]は入力ｊ，ｊ＋１，ｊ＋２を用い、一方、ｊが３２以上では、出力[ｊ]は入力ｊ＋１，ｊ＋２，ｊ＋３を用いる。

【０１７８】好ましい実施例は、ｉＭＸ１２４とＤＳＰ１２２との間で二次元アレイ（画像）のサイズを調整するためにフィルタリング計算を分割し、メモリの使用を以下のように制限する。最初に、ｉＭＸ１２４は、６４バンクの係数で３タップ行フィルタリングを行い、続いて、６４バンクの係数で３タップ列フィルタリングを行う。まず、行フィルタリングを考慮する。ｉＭＸ１２４での３タップ行フィルタリングは次の入出力関係を有する。

【０１７９】

【表２】

iMX出力	入力ポイント
0	0,1,2
1	1,2,3
2	2,3,4
…	…
31	31,32,33
32	32,33,34
33	33,34,35
…	…
61	61,62,63
62	62,63,64
63	63,64,65
64	64,65,66
…	…

【０１８０】この表と先の６３／６４サイズ調整表とを比べると、唯一の違いは、ｉＭＸが一つ余分のポイントすなわちＩＰＰ__ｏｕｔｐｕｔ[３２]を生成するものであるということが示されている。このように、好ましい実施例は、ｉＭＸ１２４で６４出力ポイントを生成し、続いて、６３有効ポイントを選ぶためにＤＳＰ１２２を用いる。

【０１８１】

【数４７】output[j]＝IPP__output[j]　　j=1, 2, …, 31のとき

IPP__output[j+1]　　j=32, 33, …, 62のとき

【０１８２】一般に、Ｎ／Ｍが１未満のときのＮ／Ｍサイズ調整は、Ｍ出力ごとのＭ－Ｎ出力の削除を含む。こうして、好ましい実施例は、一般に、ｉＭＸのような加速器においてＭ入力ポイントでのフィルタ動作を行い、続いて、ＤＳＰのようなプロセッサを用いて不必要な出力を廃棄する。（ｉＭＸはまた、Ｎ／Ｍ＝３までの１単位より大きなサイズ調整を取り扱うこともできる。）

【０１８３】ｉＭＸは、３サイクルで３タップ行フィルタの８出力を生成できる。基本的には、８つの隣り合う出力は、８ＭＡＣユニットを用いて並列に計算される。時間０では、入力ポイント０，１，２，３，．．．７を引き出し、適当な係数（各々異なり得る）と乗算して８つのアキュミュレータに累積する。時間１では、入力ポイント１，２，．．．８を引き出して同じことを行い、時間２では、入力ポイント２，３，．．．９を引き出して積を累積し、８つの出力ｊ＝０，１，．．．７を書き出す。続いて、８つの入力ポイントにわたってシフトし、ｊ＝８，９，．．．１５を計算する。垂直方向については、ｉＭＸは８つの出力を同じように並列に計算する。これらは、８つの水平に隣接する出力ポイントであって、入力アレイのすべてのフェッチも、８つの水平に隣接する出力ポイントをひと括りにする。したがって、すべての８ＭＡＣユニットは、各サイクルについて同じ係数値を共有する。垂直方向については、ｉＭＸでのデータ再使用はより少なく、それで、入出力メモリの衝突が４サイクル／８出力に計算をスローダウンする。フィルタリング時間の合計は、７サイクル／８出力または出力当り７／８サイクルである。入力データは、サイズが３２０×２４０×３である。このように、ｉＭＸのフィルタリングは、ｉＭＸが１２０ＭＨｚで走るときには３２０×２４０×３×７／８であり、２０１，６００サイクルまたは１．７ミリ秒かかる。

【０１８４】フィルタリングの後に、ＤＳＰは正しい出力を選ぶ。基本的には、６４行ごとに１行および６４列ごとに１列が廃棄される。ＤＳＰアセンブリ・ループは、有効なｉＭＸ出力ポイントを別々の出力領域に動かす。充分なローカル・メモリが双方のためにあるならば、ｉＭＸとＤＳＰとは並列して走行する。入力画像全体は大きすぎてローカル・メモリに入らないように思える。自然な選択でさえ、６３×６３出力ポイントで大き過ぎる。そのような場合は、幅６３×高さ１６に画像を分割して垂直方向に余分な簿記（bookkeeping）を取り扱う。３×６４＝１９２の係数だけであれば、前もって計算してそれらを記憶するのが経済的である。ＤＳＰは、各処理ブロックの位相の経過を追い、ｉＭＸを係数の正しい開始アドレスに向ける。カラーがインターリーブされるならば、これはインターリーブされたフィルタリングも可能とする。ｉＭＸは入力ポイントを得るのにストライド（strides）を取り扱う。以下の表は、インターリーブされた３タップ・フィルタリングを示す。

【０１８５】

【表３】

37

j	入力ポイント
0	0,3,6
1	1,4,7
2	2,5,8
...	...

【０１８６】しかしながら、インターリーブするには、各カラーにつき同じ出力ブロックサイズ用に３倍より多くメモリを費やす。このように、タスクを各カラー平面上で６３×５のようなより小さなサイズに分割し、また、垂直方向で余分のオーバーヘッドを取り扱うことが可能である。カラーフォーマットが４：４：４ではなく（例えば、４：２：２）かつ入力がカラーインターリーブされているならば、ＤＳＰはカラー平面を分けるのに更なる時間を費やす必要がある。

【０１８７】ＤＳＰ１２２において全体的にサイズ調整を行うには、直接分数アドレシングで実行されるならば、時間が費やされる。好ましい実施例は、フィルタ係数を再オーダーしダミー・ワードで膨らますことを要求することによって、計算を簡素化する。ｉＭＸ１２４は、ＤＳＰ１２２が係数を計算するのと同時に主処理を行う。これは高スループット・サイズ調整を効率的に実現する。さらに詳しくは、好ましい実施例は、ｉＭＸ１２４を用いてM位相のＫタップ・フィルタリング（冗長の出力ポイントを生成する）を行い、また、ＤＳＰ１２２を用いて正しい出力ポイントを選択することによって、画像のＮ／Ｍサイズ調整を行う。また、ＤＳＰ１２２は、メモリ使用を８＊Ｋに低減するのに、より少ないサブサンプル係数テンプレートから必要な係数を計算する。そうでなければ、２＊Ｍ＊Ｋ係数ワードまでのメモリ使用が必要となる。ＤＳＰ１２２は、係数用のおよその位置を計算し、ｉＭＸ１２４用の係数メモリを築き上げることができる。

【０１８８】広くて短い画素のブロック（すなわち、１６×６４）を処理するために、水平方向は、水平係数が垂直係数よりもより頻繁に更新されるという点で、より多くの計算が必要となる。しかしながら、ＤＳＰ１２２によって構築された係数は、短いブロック内で何度も再使用することができ、それにより、ＤＳＰ１２２上の負荷は過剰ではない。

【０１８９】とりわけ、好ましい実施例は、３タップ・フィルタおよび１０から９へのサイズ調整（例えば、３０フレーム／秒における３２０×２４０から２８８×２１６へのサイズ調整）（４：２：２または４：１：１について４：４：４にインターリーブされるものとしてサイズ調整の後にサブサンプリングを行う）について、図４２ａから図４２ｅに図示される以下のステップで進行する。１．入出力パターンを選ぶ：１０入力ごとに図４２ａのとおり９つの出力が導入される。２．処理ユニットについて係数パターンをあるカラーで最初に描く。入

38

力ポイントを指示する図４２ｂにおける矢印が用いられる：結び付けられた矢印は同じ出力ポイントを形成し、また、灰色（オープン・ヘッド）の矢印はゼロ係数を指示する。こうして、３つの入力ポイントが第１の出力ポイントを決定し、２つの入力ポイントのみが次の８つの出力ポイントの各々と９番目の無視される出力（非ゼロ入力ポイントはない）とを決定し、これが１０ごとに繰り返される。このパターンは、多位相３タップ・フィルタの使用を示唆し、また、１０の出力のグループごとに最後の出力をドロップする。３．インターリーブされた入出力を考慮する。図４２ｃを参照すると、インターリーブされた１０の入力ポイントの３つのグループの組が示されており、１０の入力ポイントの原第１のグループから第１の出力ポイントを決定する３つの入力ポイントが位置１，４，７に現在あり、１０の入力ポイントの原第２のグループから第１の出力ポイントを決定する３つの入力ポイントが位置２，５，８に現在あり、１０の入力ポイントの原第３のグループから第１の出力ポイントを決定する３つの入力ポイントが位置３，６，９に現在あり、以下同様であるものとされる。このインターリーブは、３つの隣接出力ポイントの組がすべての異なる入力ポイントを用いるとともに同時メモリアクセスを必要としないということを意味する。４．８重の並列処理とｉＭＸとを考慮し、必要ならばさらにダミーの出力を加える。図４２ｄを参照すると、並列計算用の８からなる４つのグループに分割された出力ポイントが示されている。５．グループ分けされた係数およびオーダーを計算する。ｉＭＸは、左から右へのそして上から下への係数オーダーを用いて１度に１つのグループを処理し、その後、次のグループへと進む。係数は同じオーダーに整列される必要がある。ｉＭＸ係数メモリとフラッシュメモリとがこれらすべての係数を収容できるならば、これらの係数はＤＳＰコードに一定のデータとして含むことができ、また、このステップはソフトウェア開発において１度に行われる。ｉＭＸ係数メモリがこれらの係数をずっと保持することができるがこれらがフラッシュメモリにおいてあまりにも大きな場所を占めるならば、このステップはシステムの初期化の間に１度行われる。同様に、ＳＤＲＡＭはこれらのすべての係数を保持できるが、ｉＭＸ係数メモリはそれらをずっと保持できない。このステップはシステムの初期化において１度行われ、また、係数画像はＳＤＲＡＭに記憶される。必要なときは、これらの係数はＳＤＲＡＭから交換される。これらすべての係数をいかなるときにも記憶するのは望ましくなく、とりわけ、Mが非常に大きい（１００＋）ときは、係数の必要な「ウインドウ」をｉＭＸ処理と同時にＤＳＰで計算する。ｉＭＸ係数メモリが計算ブロックのために必要な係数を保持できることを確認だけすること。６．ｉＭＸで計算を始める。この場合、２７有効出力ポイントを生成するのにインナーループで約１２サイ

クル掛かる。各ｉＭＸコマンドは、２－Ｄ出力ブロックを生成でき、それにより、１６×２７出力ポイントを生成するのに約１０＋１６＊１２＝２０２サイクル掛かる。７．ｉＭＸが行われたとき、ＤＳＰに正しい出力ポイントを選ばさせる。この例では、２７６ポイントが３２出力ポイントの全グループから選ばれる。このタスクは、出力の幅が３＊Ｍにマッチするかその倍数であるならば、符号化するのがより簡単である。ＤＳＰは、各有効出力に１度触れることのみしなければならず、それにより、ＤＳＰのローディングは重要なものとはならない

【０１９０】垂直サイズ調整では、ｉＭＸはＳＩＭＤモードで働く。８つの隣接データ入力の各グループが並列に処理される。係数はサイクルごとに１つの値で用いられ、この値はすべてのカラー成分に適応される。サイズ調整ファクターが水平および垂直で同じであるとしても、どのようにｉＭＸが係数を用いるかは異なっており、それにより、別々の垂直サイズ調整係数記憶である（それには水平係数の１／３掛かる）必要がある。図４２ｅを参照のこと。ここでも、ｉＭＸにおけるすべての垂直係数をスワップ・インおよびスワップ・アウトに保つか、ＤＳＰに休む間もなく計算させるというオプションがある。ＤＳＰは、ｉＭＸが処理を完了した後に、有効出力行を選ぶ必要がある。

【０１９１】色調スケーリングの好ましい実施例

色調スケーリングは、細部をより明瞭にするために、画像の輝度信号（または、カラー信号）のダイナミックレンジについて作動する。例えば、逆光でまたは非常に明るい環境で撮られた写真は、典型的には、高い明るさレベルを有する。色調スケーリングは、通常、図４３によってブロック形状で図示される輝度（または、カラー）ヒストグラム均一化に依存する。実際、コンバータ・ブロック４３０は、入力輝度レベルを（８ビットについては０～２５５の範囲において、または、１２ビットについては０～４０９５の範囲において）ルックアップ・テーブルを用いて同じ範囲における出力輝度レベルに変換する。ルックアップ・テーブルは、入力レベルと、ヒストグラム均一化ブロック４３２において以下のように計算される出力レベルを有する対応出力レベルとのペアである。まず、色調スケーリングが適応される画像の入力輝度レベルの累積分布関数を求める。すなわち、Ｆ（ｒ）＝（レベルがｒ以下である画素の数）／（画像における画素の合計数）であるようなＦ（ｒ）を求める。次に、Ｆ（ｒ）を最大画素レベルと乗算し最も近い整数に四捨五入することにより、ルックアップ・テーブル関数Ｔ（ｒ）を生成する。そのとき、ルックアップ・テーブルは、ｓ＝Ｔ（ｒ）であるレベル（ｒ，ｓ）のペアにすぎない。図４５は、暗部の細かな細部は認識するのが困難である現像下の画像（画素の大半は、小さなｒについてのＴ（ｒ）の大きなスロープによって反射されるので、低レベルを有する）についてのＴ（ｒ）を例示する。また、この現像下の画像については、図４５に示されるように、色調スケーリングはレベルｒ＝５００をｓ＝２０００に変換し、したがって、色調スケーリングされた画像においては、輝度レベルの相違が低レベルについては強調され高レベルについては強調されない。こうして、色調スケーリングは暗部における細部を強調する。

【０１９２】しかしながら、色調スケーリングされた画像は、カラーが明瞭でありすぎて、色調スケーリングされた画像が油絵で描かれたものであるかのように、不自然に見える。このように、医療やナイト・ビジョンのような他の応用は不自然ながらも細かい細部を要求するけれども、この色調スケーリングは、たとえ細かな細部がより明瞭であっても、不自然な特徴のために消費者用には強すぎるものとなることがある。

【０１９３】好ましい実施例は、ヒストグラム均等化関数Ｔ（ｒ）と原画像レベルｒとの線形組合せを用いることによって色調スケーリングを提供する。すなわち、０以上１以下のパラメータαについて、色調スケーリング関数は次式で定義される。

【０１９４】

【数４８】ｓ＝Ｒｏｕｎｄ（αＴ（ｒ）＋（１－α）ｒ）

【０１９５】ここで、続くαとの乗算，（１－α）ｒとの加算および四捨五入のために、最も近い整数に四捨五入することがＴ（ｒ）の定義においては必要でないという点を除いて、Ｔ（ｒ）は先に説明したとおりである。図４５は、曲線ｓ＝Ｔ（ｒ）と恒等線ｓ＝ｒとの間のα＝０．３についての好ましい実施例を例示する。

【０１９６】図４４は、機能ブロック形状における好ましい実施例の色調スケーリングを示し、ここでも、ブロック４４２において輝度（または、カラー）レベルについてヒストグラム均等関数Ｔ()を定義し、Ｔ()の重みαとブロック４４４における恒等式との四捨五入された線形組合せを定義して、コンバータ４４０における色調スケーリング用の最終ルックアップ・テーブルを得る。重みαが０に等しいときは、色調スケーリングはなく見た目も自然ではないが、重みαが１に等しいときは、色調スケーリングがＴ()で行われ細かい細部が強調される。重みαの値は応用に従って選択できる。計算のすべてはプログラム可能である。

【０１９７】実施の詳細

先の機能を支援する好ましい実施例のハードウェア構造は以下のものを含む。

【０１９８】ＳＤＲＡＭコントローラ

ＳＤＲＡＭコントローラ・ブロック１１０は、ＳＤＲＡＭ１１０とプロセッサ（ＡＲＭ１３０，ＤＳＰ１２２），ＣＣＤコントローラ１０２，ＴＶ符号器１０６，プレビュー・エンジン１０４などのようなすべての機能ブロックとの間のメイン・インターフェイスとして動作する。それは８０ＭＨｚまでのＳＤＲＡＭタイミングを支援する。それはまた、連続データ・アクセス用の低オ

ーバーヘッドを提供する。それはまた、ＣＣＤデータ入力およびＴＶ表示データ出力の実時間データストリームを支援するためにアクセスユニットを優先付ける能力を有する。それはまた、外部ＳＤＲＡＭ用のパワーダウン制御を提供する。ＤＳＰ１２２は、データ・アクセスがない間にＳＤＲＡＭ１６０のＣＫＥ信号を禁止することができる。ＳＤＲＡＭコントローラ・ブロック１１０は、１６／６４／１２８／２５６ＭＢ・ＳＤＲＡＭと、３２ビット幅または２×１６ビット幅ＳＤＲＡＭと、最大８０ＭＨｚ（例えば、１０～８０ＭＨｚ）動作と、ワード，ハーフ・ワードまたはバイト・アクセス（ＡＲＭ）の利用可能性とを支援し、モード設定とパワーダウンおよび自己リフレッシュ・プログラム可能リフレッシュ間隔とをコマンドし、２または３のＣＡＳ待ち時間を＊選択可能とすることができ、２チップ選択出力（最大ＳＤＲＡＭサイズが１Ｇビット）をコマンドし、ＤＭＡ転送を許可および管理し、プロセッサとＳＤＲＡＭとの間の、ＣＣＤデータバッファからＳＤＲＡＭへの、プレビュー・エンジンからＳＤＲＡＭへの、バースト圧縮からＳＤＲＡＭへのまたはその反対の、ＳＤＲＡＭからビデオ符号器への、ＳＤＲＡＭからＯＳＤへの、ＡＲＭからＳＤＲＡＭへのまたはその反対の、ＤＳＰ画像バッファからＳＤＲＡＭへのまたはその反対の、データ・フローを管理する。図１２ａはＳＤＲＡＭコントローラによって管理されるデータ・フローを示す。信号および優先順位は以下のとおりである。

【０１９９】

【表４】

信号名	信号の説明
Clk	ＳＤＲＡＭクロック（１０～８０ＭＨｚ）
Req	データ読出し／書込み要求信号
req__en	ＳＤＲＡＭコントローラからの要求イネーブル（アクノリッジ）信号 周辺モジュールがデータＩＮ／ＯＵＴを要求するとき、ｒｅｑ信号がアサートされ、ｒｅｑ__ｅｎ信号がアサートされるとき、ｒｅｑ信号は無効とされる。
Address	読出しまたは書込みのアドレスを開始する。 ＣＣＤＣ，ＰＲＥＶＩＥＷ，ＢＵＲＳＴＣ，ＥＮＣ，ＯＳＤ，ＤＳＰ：２２ビット幅 ＡＲＭ：２５ビット幅
Odata	ＳＤＲＡＭ（３２ビット）にデータを出力
Idata	ＳＤＲＡＭ（３２ビット）からデータを入力
Rw	読出しまたは書込み信号 ０：書込み　／　１：読出し
Dten	ＤＳＰ　ＩＦ用のデータ書込みイネーブル信号
Ds	ＡＲＭ　ＩＦ用のバス選択（４ビット）

【０２００】アクセス・ユニットの優先順位表は以下のとおりである。

【０２０１】

【表５】

優先順位	アクセス・ユニット
１（最も高い）	ENC out
2	CCD in
3	OSD out
4	PRVW in
5	BURST in
6	DSP I/O
7	ARM I/O

【０２０２】プレビュー・エンジン

図１４は、ＣＣＤ生データからの４：２：２フォーマットのＹＣｂＣｒをＣＣＤコントローラ１０２から画像データに提供するとともに以下のメイン機能を有する好ましい実施例のプレビュー・エンジン１０４のブロック図である。ＲＧＢ・ＣＣＤおよび相補（ＹｅＣｙＭｇＧ）ＣＣＤの双方について利用可能である（図７ａおよび図７ｂがこれらのＣＣＤパターンを示す）

デジタル利得調整

ホワイトバランス

垂直および水平ノイズフィルタ

相補ＣＣＤ用のＲＧＢ利得調整

ＲＧＢカラー用の独立ガンマ補正

ＹＣｂＣｒ－４：２：２でフォーマットされたデータ出力

【０２０３】同期モジュール１４０２は、画像の開始ポイント用の同期信号およびダウンサンプリング用のイネーブル信号のような他のモジュール用の制御信号を生成する。このモジュールでは、画像処理は行われない。ホワイトバランス・モジュール１４０４は、ＣＣＤ生データについてデジタル利得調整およびホワイトバランスを行う。ＣＦＡ補間モジュール１４０６は、水平ノイズフィルタ，水平補間，垂直ノイズフィルタ，垂直補間，ダウンサンプリングなどのような多くの重要なサブモジュールを有する。このモジュールは、ＣＣＤモード（ＲＧＢ　ＣＣＤまたは相補ＣＣＤ）に関係なくＲＧＢフォーマットされたデータを出力する。相補ＣＣＤ用のＲＧＢ利得モジュール１４０８は、相補ＣＣＤ用のＲＧＢカラー・フォーマットによってホワイトバランスを行う調整を可能とする。ガンマ補正モジュール１４１０は、４つの線形セグメントを有する近似ガンマ曲線でガンマ補正を行う。このモジュールは、各カラーについて存在してＲＧＢへの独立調整を可能とする。ＲＧＢ２ＹＣｂＣｒ変換モジュール１４１２は、ＲＧＢフォーマットされたデータをＹＣｂＣｒフォーマットされたデータに変換し、ＣｂおよびＣｒのオフセットを調整する。４：２：

２変換モジュール１４１４は、ＹＣｂＣｒ－４：４：４フォーマットされたデータを２：２：２フォーマットに変換し、それらを３２ビット・データ・バスに出力する。ＳＤＲＡＭインターフェイス・モジュール１４１６は、ＳＤＲＡＭコントローラ１１０（図１ｂ）と通信し、それにＹＣｂＣｒ－４：２：２フォーマットされた画像データを記憶するように要求する。

【０２０４】以下にモジュールを説明する。ホワイトバランス・モジュール１４０４は、ＣＣＤ生データについてデジタル利得調整およびホワイトバランスを行う。デジタル利得は画像の全体明るさを調整し、ホワイトバランスはＣＦＡパターンで存在するカラーの比を調整する。図８は、ホワイトバランス・モジュール１４０４のブロック図である。２つの利得調整のために２つの乗算器があり、また、回路のサイズを低減するためにクリップ回路がある。この図でＰＶＧＡＩＮと名付けられたデジタル利得用の利得値はＰＶＧＡＩＮレジスタのデータを用い、また、ホワイトバランスはＣＦＡパターン・レジスタを設定することによって自動的に選択される。

【０２０５】ＣＦＡ補間モジュール１４０６は、水平および垂直補間，水平および垂直ノイズフィルタリング，ダウンサンプリング，カラー調整およびＲＧＢカラー変換への相補カラー用のサブモジュールを含む。図１０ａは、ＣＦＡ補間モジュール１４０６のブロック図である。水平ノイズフィルタ・サブモジュール１００２は、３タップ・ローパスフィルタ水平フィルタを行う。図１０ｂを参照。水平補間フィルタ・サブモジュール１００４は、２種類のフィルタを用意しており、それらの１つを用いて水平に補間する。出力信号「Ｌ」および「Ｒ」はライン上の左データおよび右データを意味する。例えば、処理されたラインは次のＣＦＡパターンＧＢＧＢＧＢＧＢＧＢ．．．で始まり、出力信号「Ｌ」はＧであり、出力信号「Ｒ」はＢである。したがって、これらの２つの出力は各ラインでカラーを変える。水平ダウンサンプリング・サブモジュール１００６は、水平デシメーション・パターンのレジスタ設定に基づいて有効画素上のデータのみを出力する。垂直補間サブモジュール１００８は、プレビュー・エンジン・モジュールの外で２つのラインメモリ１０１０を用いて３タップ垂直補間フィルタを処理するとともに、ＣＦＡパターンに存在するすべてのカラーのデータを出力する。そして、このサブモジュールは垂直ノイズフィルタも実行する。カラー選択サブモジュール１０１２は、ＣＦＡパターンでカラーごとにデータを抽出するとともに、ＲＧＢカラー・フォーマットされたデータをＲＧＢ ＣＣＤモードで出力するか相補カラー・フォーマットされたデータを相補ＣＣＤモードで出力する。この図では、「ｇ」信号は、Ｇに関する一時データであり、次のカラー調整サブモジュール１０１４においてＲおよびＢを再計算するのに用いられる。カラー・フォーマットされたデータはカラー調整サブモジュール１０１４でカラー調整処理され、また、その処理はＣＣＤモードによって異なる。垂直補間サブモジュール１００８からカラー調整サブモジュール１０１４へのこの画像処理は、ＣＣＤモードおよび垂直補間モードに依存して強い相関関係を有する。したがって、その処理は、以下で説明される一連の垂直補間処理として考慮される。Ｃｏｍｐ２ＲＧＢ変換サブモジュール１０１６は、相補ＣＣＤモードにおいて補色フォーマットをＲＧＢカラー・フォーマットに変換する。ＲＧＢ ＣＣＤモードでは、データはこのサブモジュールを迂回する。

【０２０６】以下のセクションはこれらのサブモジュールを説明する。水平ノイズフィルタ１００２は、３タップ水平ローパスフィルタを実行し、ランダムノイズを効率的に低減することができる。実際、データの中心がX_0に設定されるとき、ＣＦＡパターンおよびそれの処理されたラインに依存して以下の計算が実行される。

【０２０７】

【数４９】

$$X_0=\begin{cases}(X_{-2}+2X_0+X_2)\ /4 & \text{（処理ラインに２つのカラー）}\\ (X_{-1}+2X_0+X_1)\ /4 & \text{（処理ラインに１つのカラー）}\end{cases}$$

【０２０８】このフィルタのオン／オフ・スイッチングはレジスタ設定によって制御できる。

【０２０９】図１０ｂは、水平ノイズフィルタ・サブモジュール１００２のブロック図である。２種類のフィルタが、２つの加算器とこの図で「three__taps__sw」と名付けられたスイッチとを用いて実行される。処理ラインに１つのカラーがあるならば、スイッチはオンに設定される（図においてハイ）。このスイッチは、ＣＦＡパターンのレジスタ設定と処理画像のラインの位置とに依存して自動的に制御される。出力の前に、ノイズフィルタリングされたデータまたは迂回されたデータがレジスタ設定によって選択される。

【０２１０】水平補間サブモジュール１００４では、２つのモードのフィルタリングがあり、水平ノイズフィルタ１００２からのデータは２タップか５タップかのいずれかの補間フィルタによって水平に補間される。２タップ・フィルタは、中央のデータを補間するために、左および右の隣接画素での２つのデータの平均を用いる。このモードは「単純モード」と呼ばれる。５タップ水平補間フィルタは、処理画像におけるエッジの周りの誤りカラーを効率的に低減することができるように、処理ライン上で別のカラーの情報を使用する。このモードは「通常モード」と呼ばれる。これらのモードはレジスタ設定によって選択可能である。実際、データの中央がX_0に

45 46

設定されるとき、補間モードに依存して以下の計算が実行される。

$$X_0=\begin{cases}(-X_{-2}+2X_{-1}+2X_0+2X_1-X_2)\ /\ 4 & \text{(通常モード)}\\ (X_{-1}+X_1)\ /\ 2 & \text{(単純モード)}\end{cases}$$

【０２１２】図１０ｃは、ＲＧＢバイエルＣＣＤモードにおけるこの水平補間処理の一例を示す。この図において、補間されるデータは小文字によって表されている。

【０２１３】図１０ｄは、水平補間モジュール１００４のブロック図である。２つの加算器，１つの減算器およびフィルタ・モード・スイッチが、これら２種類のフィルタの１つを実行するために実現される。フィルタ・モード・スイッチは、レジスタを設定することによって制御される。

【０２１４】垂直補間サブモジュール１００８は、プレビュー・エンジン・モジュールの外の２つのラインメモリを用いて２タップか３タップかのいずれかの垂直補間フィルタを処理し、ＣＦＡパターンに存在するすべてのカラーの情報を出力する。そして、このサブモジュールは垂直ノイズフィルタも実行する。このモジュールにおける画像処理は少し複雑であり、このサブモジュールからの出力は処理ライン，ＣＣＤモード，ＣＦＡパターン，フィルタ・モードおよびノイズフィルタ・オン／オフによって変化する。以下で説明されるように、垂直補間サブモジュール１００８からカラー調整サブモジュール１０１４までの画像処理は強い相関関係を有してお※り、それらのこの処理フローは一連の垂直補間処理と考えられる。したがって、この一連の垂直補間処理がまず説明される。そのシーケンスは「垂直補間シーケンス」と呼ばれる。

【０２１５】水平補間のように、垂直補間処理も、２種類の補間モードすなわち「単純モード」と「通常モード」とを有する。単純モードにおける補間フィルタは、上および下の隣の画素での２つのデータの平均を使用してデータの中央を補間する。通常モードでは、処理はＲＧＢ ＣＣＤモードと相補ＣＣＤモードとの間で異なる。ＲＧＢ ＣＣＤモードにおける通常モードでの補間フィルタは、水平補間フィルタと同じ他のカラーの１つのデータを使用する。実際、補間されるべきあるカラーのデータがＸ（主に、Ｒ，Ｂ）に設定されるとともに基準として用いられるカラーのデータがＹ（主に、Ｇ）に設定されるとき、この垂直補間シーケンスを通して補間モードに依存して以下の計算が実行され、そして、それはカラー調整サブモジュールからの出力である。

【０２１６】

【数５１】

$$X_0=\begin{cases}(X_{-1}-Y_{-1}+X_1-Y_1)\ /\ 2+Y_0 & \text{(通常モード)}\\ (X_{-1}+X_1)\ /\ 2 & \text{(単純モード)}\end{cases}$$

【０２１７】図１０ｅは、ＲＧＢバイエルＣＣＤパターンについてのこの垂直補間シーケンスの一例を示す。

【０２１８】相補ＣＣＤモードでは、通常モードとは「カラー調整のある単純補間」を意味する。すなわち、単純垂直補間によって処理されるすべてのカラーのデータが、補色空間における公式に基づいて調整される。実★際、補間されるべきあるカラーのデータがＸに設定され、他のカラーのデータがＷ，ＹおよびＺに設定されるとき、相補ＣＣＤモードにおける通常モードでは以下の計算が実行される。

【０２１９】

【数５２】

$$X_0=\begin{cases}(X_{-1}+X_1)\ /\ 2\pm a(w_0,x_0,y_0,z_0) & \text{(通常モード)}\\ (X_{-1}+X_1)\ /\ 2 & \text{(単純モード)}\end{cases}$$

【０２２０】ａ＝ａ（w_0，x_0，y_0，z_0）の計算については、以下を参照のこと。

【０２２１】この垂直補間シーケンスでは、垂直補間サブモジュール１００８の主な役割は垂直補間シーケンスの一部および垂直ノイズフィルタを実行することである。垂直補間シーケンスの一部とは通常の垂直補間モード用のデータを準備することを意味する。図１０ｅおよび図１０ｂに示されるように（それぞれＲＧＢおよび相補ＣＣＤパターンについて）、単純モードでは、この垂直補間サブモジュールの出力データはカラー調整サブモジュールを迂回する。したがって、単純モードでは、このサブモジュールからの出力が垂直補間シーケンスの出力として用いられる。補間モードのいずれの場合においても、このサブモードは垂直補間シーケンスについて以下の等式を計算する。

【０２２２】

【数５３】$X_0=(X_{-1}-X_1)\ /\ 2$

【０２２３】次の３タップ垂直ローパスフィルタを実行する垂直ノイズフィルタはまた、ＣＦＡパターンに依存してこのサブモジュールで処理される。

【０２２４】

【数５４】$X_0=(X_{-1}-2X_0-X_1)\ 4$

【０２２５】しかしながら、このフィルタリングについて、処理される３つのライン上の同じカラーのデータが準備されなければならない。したがって、垂直ノイズフィルタの機能は、主に、ＲＧＢバイエルＣＣＤにおいて

Ｇのみを実行する。図１０ｇは、ＲＧＢバイエルＣＣＤについてのこの垂直補間サブモジュールの出力の一例を示す。垂直ノイズフィルタが適用できてかつそれがオンに設定されるとき、原データ（この図ではＲ）も、他のカラー（この図ではＧ）との相関関係を保つために調整される。

【０２２６】図１０ｈは、垂直補間サブモジュール１００８のブロック図である。垂直補間およびノイズフィルタリングを実行するために、６つの加算器と２つの減算器とが実現される。特に、Ｌ＿１２１およびＲ＿１２１の計算処理は非常に複雑であって、Ｌ＿１２１およびＲ＿１２１用のスイッチング動作はこの図を単純化するために示されていない。カラー選択サブモジュール１０１２は、カラー・フォーマットのオーダーに、すなわち、ＲＧＢ ＣＣＤモードにおいてはＲ，ＧおよびＢに、相補ＣＣＤモードにおいてはＹｅ，Ｃｙ，Ｍｇ，Ｇに、垂直補間サブモジュールからの入力を整列する。この整列はＣＦＡパターンのレジスタを設定することによって自動的に実行される。図１０ｉは、図１０ｇのＲＧＢバイエルＣＣＤにおけるこのカラー選択処理の一例を示す。この図において「ｇ」と名付けられた出力は、Ｇの一時データであって、カラー調整サブモジュールにおけるＲＧＢ ＣＣＤモードでのＲまたはＢの再計算のために用いられる。

【０２２７】図１０ｊは、カラー選択サブモジュール１０１２のブロック図である。４つのカラー抽出器が、垂直補間サブモジュール１００８からの４つの入力から正しいカラーに独立して切り替えて選択する。カラー調整サブモジュール１０１４は垂直補間シーケンスについての残りの計算を行う。ＲＧＢバイエルＣＣＤのようなＲＧＢ ＣＣＤモードでは、ＲまたはＢはＧの一時データを用いて再び計算される。カラー選択サブモジュールからのＲまたはＢのデータがＸに設定されるとき、ＲＧＢ ＣＣＤモードにおいて以下の計算が行われる。

【０２２８】

【数５５】$x = X - G_{temp} + G$

【０２２９】図１０ｉの例では、ノイズフィルタがオフであるとき、

【０２３０】

【数５６】$X = (b_{02} - b_{22}) / 2$

$G_{temp} = (G_{02} + G_{22}) / 2$

$G = g_{12}$

したがって、

$x = B$

$= (b_{02} - b_{22}) / 2 - (G_{02} + G_{22}) / 2 + g_{12}$

$= ((b_{02} - G_{02}) + (b_{22} - G_{22})) / 2 + g_{12}$

【０２３１】これは、カラー調整モジュールの出力Ｂであり、また、垂直補間シーケンスの出力でもある。すなわち、ＲＧＢ ＣＣＤモードにおける垂直補間シーケンスは、補間されるべきカラーのデータと他のカラーの基準データとの間の差の平均を用いる。

【０２３２】相補ＣＣＤモジュールでは、カラー調整はカラー選択サブモジュールからのすべてのデータに対して処理される。まず、値ａが、相補カラー空間における公式Ｙｅ＋Ｃｙ＝Ｇ＋Ｍｇに基づいて各画素で計算される。

【０２３３】

【数５７】ａ＝Ｇ＋Ｍｇ－Ｙｅ－Ｃｙ

【０２３４】すなわち、値ａは４つのカラーのエラー値の量と考えることができる。したがって、相補ＣＣＤモードでは、すべてのカラーＹｅ，Ｃｙ，Ｍｇ，Ｇに対して上記公式を満たすために、以下の調整が処理される。

【０２３５】

【数５８】ｙｅ＝Ｙｅ＋ａ／４

ｃｙ＝Ｃｙ＋ａ／４

ｇ＝Ｇ－ａ／４

ｍｇ＝Ｍｇ－ａ／４

【０２３６】図１０ｋは、カラー調整サブモジュール１０１４のブロック図である。上述した２種類の計算を行うために、６つの加算器と３つの減算器とが実現される。この図において、ＣＣＤＭＯＤと名付けられたスイッチャーが、ＣＣＤモードに依存して正しい出力を選択し、レジスタを設定することによって制御される。

【０２３７】ｃｏｍｐ２ＲＧＢ変換サブモジュール１０１６は、相補ＣＣＤモードでは相補カラー・フォーマットされたデータをＲＧＢフォーマットされたデータに変換する。特に、Ｇについては、カラー調整からのデータおよび変換公式によって計算されるデータは、５種類の混合比で混合することができる。実際、以下の計算が変換公式に基づいて行われる。

【０２３８】

【数５９】Ｒ＝Ｙｅ－Ｃｙ＋Ｍｇ

$G = r G_{input} + (1 - r)$（Ｙｅ＋Ｃｙ－Ｍｇ）

（ｒ＝０，１／４，２／４，３／４，１）

Ｂ＝Ｍｇ－Ｙｅ＋Ｃｙ

【０２３９】ＲＧＢ ＣＣＤモードでは、カラー調整サブモジュールからのデータはこのサブモジュールを迂回する。

【０２４０】図１０ｌは、ｃｏｍｐ２ＲＧＢ変換サブモジュール１０１６のブロック図である。上の計算を行うために、３つの加算器と３つの減算器と２つの乗算器とが実現される。この図において「green_ratio」と名づけられたＧについての利得調整器はレジスタを設定することによって調整可能である。ＲＧＢ ＣＣＤモードでは、ＣＣＤＭＯＤスイッチャーは、このモジュールを迂回するためにオフ（この図ではハイ）を選択する。相補ＣＣＤモジュールについてのＲＧＢ利得は、相補ＣＣＤモジュールのためでさえあるＲＧＢカラー・フォーマットによってホワイトバランスの調整を可能とする。このモジュールはＲＧＢ ＣＣＤモードにおいても利用でき

る。

【０２４１】図９ａは、相補ホワイトバランス・モジュール１４０８のブロック図である。この動作のために、１つの乗算器とクリップ回路とが実現される。ＲＧＢについての各利得がレジスタによって設定される。ガンマ補正モジュール１４１０は、ＲＧＢカラー・フォーマットで各カラー・データについてガンマ補正を行う。この動作のために、４つの線形セグメントでガンマ曲線を近似するために、前もって３種類のデータを用意する。これらは、図９ｂに示される領域，オフセットおよび利得である。図１４に示されるように、このモジュールは、ＲＧＢへの独立調整が行われるように、各カラーについて存在する。図９ｃは、ガンマ補正モジュール１４１０のブロック図である。領域検出器は、領域データに基づいて入力データ用の正しい利得およびオフセットを選択する。利得，オフセットおよび領域に関するデータは３つのレジスタで設定される。

【０２４２】ＲＧＢ２ＹＣｂＣｒ変換モジュール１４１２は、ＲＧＢフォーマットされたデータをＹＣｂＣｒフォーマットされたデータに変換し、以下の行列計算に基づいてＣｂおよびＣｒに対してオフセットを調整する。

【０２４３】

【数６０】

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} COEF1 & COEF2 & COEF3 \\ COEF4 & COEF5 & COEF6 \\ COEF7 & COEF8 & COEF9 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ OFFSET\_Cb \\ OFFSET\_Cr \end{bmatrix}$$

【０２４４】この行列の各係数は、この変換用の可変設定が利用できるように、レジスタで設定される。

【０２４５】図１１は、このＲＧＢ２ＹＣｂＣｒ変換モジュール１４１２のブロック図である。前述した行列計算のために、９つの乗算器と５つの加算器とが実現される。ＲＧＢデータを係数と乗算した後に、乗算器からの各データの最も重要でない６つのビットが、回路のサイズを低減するために、カットされる。ＣｂおよびＣｒについては、オフセット調整用の更なる回路に、ＹＣｂＣｒ変換回路が続けられる。ＣｂおよびＣｒ用のクリップ回路には、２者の相補からオフセット・バイナリへの変換回路が含まれる。

【０２４６】バースト・モード圧縮／圧縮解除エンジン

好ましい実施例のＤＳＣエンジンには、通常の捕獲モードと比較して画像の解像度において妥協のない、リアルタイム処理を行う改良されたバースト捕獲機能が含まれる。バースト捕獲モードは、増大されたバースト捕獲シーケンス長用の専用の圧縮および圧縮解除エンジン１０８を使用するものである。ＣＣＤ生画像フレームのシーケンスが、圧縮エンジン１０８を用いて、ＳＤＲＡＭ１６０にまず記憶される。その後、オフライン処理として、通常の捕獲モードの画像パイプラインが、ＣＣＤ生画像をＳＤＲＡＭ１６０から検索し、それらを連続して処理し、最後にそれらをＪＰＥＧファイルとしてＳＤＲＡＭに記憶し直す。動画再生モードはこれらのＪＰＥＧファイルを表示することができる。

【０２４７】バースト・モード圧縮／圧縮解除エンジン１０８は、差分パルス符号変調（ＤＰＣＭ）と、ベースラインＪＰＥＧ圧縮におけるＤＣ係数のエントロピー符号化と同じ表を用いたハフマン（Ｈｕｆｆｍａｎ）符号化とを含む。エンジン１０８は、クロミナンスＤＣ差分データ用のＪＰＥＧ基準におけるサンプル・ハフマン表を用いる。エンジン１０８は、図１３に例示される逆変換も提供する。固定ハフマン表（クロミナンスＤＣ係数用のＪＰＥＧハフマン表）を以下に示す。

【０２４８】

【表６】

カテゴリ (SSSS)	$\hat{D}_i$	符号長	符号語
0	0	2	00
1	-1,1	2	01
2	-3,-2,2,3	2	10
3	-7,..,-4,4,..,7	3	110
4	-15,..,-8,8,..,15	4	1110
5	-31,..,-16,16,..,31	5	11110
6	-63,..,-32,32,..,63	6	111110
7	-127,..,-64,64,..,127	7	11111110
8	-255,..,-128,128,..,128	8	111111110
9	-511,..,-256,256,..,511	9	1111111110
10	-1023,..,-512,512,..,1023	10	11111111110
11	-2047,..,-1024,1024,..,2047	11	111111111110
12	-4095,..,-2048,2048,..,4095	12	1111111111110

【０２４９】符号器は４つのルックアップ・テーブルを有する。すなわち、ハフマン符号（１３×２バイト・エ

ントリ）とハフマン符号長表（１３×１バイト・エントリ）と可変長ビットストリームを生成する低ビット・マスク（３２×４バイト・エントリ）とログ・テーブル（２５６×１バイト・エントリ）とである。ハフマン表は、簡略化のためプログラム可能ではないが、代わりの実施例はプログラム可能ハフマン表を含むこともできる。

【０２５０】ハフマン復号器は、ハフマン符号器の逆関数を行い、５つのルックアップ・テーブルを有する。すなわち、最大符号比較表（１３×２バイト・エントリ）と最小符号比較表（１３×２バイト・エントリ）と復号化ハフマン符号ポインタ（１３×１バイト・エントリ）と復号化ハフマン符号表（１３×１バイト・エントリ）とビット位置マスク（３２×４バイト・エントリ）とである。損失のあるモード圧縮は、各係数の最も重要でないビット（ＬＳＢ）または２つの最も重要でないビットを単に廃棄する。

【０２５１】再生同期

オーディオ・ビジュアル・ビットストリームを再生する上での問題は、オーディオをビデオ信号といかにして同期させるかということである。好ましい実施例は、リアルタイムでバックグラウンドにおいて継ぎ目なくオーディオ・ビットストリームを再生するが、オーディオはＩＴＵ－ＴＧ．７１１およびマイクロソフト１６ビットＰＣＭのような簡単な符号化標準を用いて符号化されている。中断サービス・ルーチンを用いることによって、ＤＳＰ源の約０．１％が（マルチチャンネルの）バッファされたシリアルポートを通してリアルタイムでオーディオを出力するのに十分である。図１ｂを参照のこと。したがって、好ましい実施例は、オーディオ再生に同期してビデオ復号化を実現しなければならない。明瞭化のため、オーディオおよびビデオの双方が最高スピードで（オーディオについては８Ｋサンプル／秒およびビデオについては３０フレーム／秒のリアルタイムで）捕獲されるとする。オーディオはサンプルとして再生される。しかしながら、ビデオはフレームの粒子において表示される。こうして、同期の問題が、ビデオ復号化がリアルタイム要請よりも早いか遅いかという事実によって引き起こされる。ビデオ復号化が早すぎるならば、ある量の遅延スロットが挿入されて、復号化をスローダウンしなければならない。反対に、ビデオ復号化が遅過ぎるならば、リアルタイム・オーディオ再生に追いつくように、いくつかのビデオ・フレームがスキップされなければならない。

【０２５２】好ましい実施例は双方の場合を取り扱う。特に、遅いビデオ復号化の場合には、好ましい実施例は最適なやり方でフレームを適切に選択およびスキップすることができる。好ましい実施例は、２方向符号化フレーム（Ｂフレーム）なしのビデオ・ビットストリームについて説明されることに注意すること。図４６ａは、オーディオとビデオとの間の同期を図示する。最初のビデオ・フレームは、オーディオ・ビデオ再生を始める前に予め復号化される。ビデオはフレームの粒子において表示されるので、同期ポイントはビデオ・フレーム境界すなわち｛ｔ＝０，ΔＴ，２ΔＴ，３ΔＴ．．．｝に位置する。ここで、ΔＴはフレームの期間であり、それは次式で定義される。

【０２５３】

【数６１】

$$\Delta T = 1 / fp \qquad (1)$$

【０２５４】ここで、ｆｐは、ビデオ・シーケンスに用いられるフレーム速度である。

【０２５５】ビデオ復号化スピードが十分に速くないとき、オーディオおよびビデオは同期を失い得る。図４６ａに例示される通り、ビデオ・フレーム２の復号化が時間内で終わらない（Ｔｄ２＞ΔＴ）とき、オーディオ・ビデオ再生は、ビデオ・フレーム１を表示した後に、同期を失う。ここで、｛Ｔｄｍ，ｍ＝０，１，２，．．．｝は、ビデオ・フレームｍを復号するのに用いられる復号化時間を示す。ビデオの再生スピードが十分でないとき、オーディオとビデオとの間の適切な同期を維持する唯一の方法はビデオ・フレームを適当にスキップすることである。図４６ｂにおいて、ビデオ・フレーム２は、同期がフレーム３で再び得ることができるように、スキップされる（かつ、フレーム１は繰り返される）。

【０２５６】好ましい実施例の循環バッファ構成が図４７に例示されている。ビデオ復号器は、循環バッファの一方の側に接続され、ディスプレイが他方の側に接続される。循環バッファは、サイズＮのビデオ・フレームを有する。循環バッファの各フレーム・バッファに関連する２つのレジスタがある。すなわち、バッファｎに記憶されるビデオ・フレームの推定表示時間を指示するＴＰ$_n$（ｎ＝０，１，２，３，．．．，Ｎ－１）を含む第１のレジスタと、バッファｎにおけるフレームが表示のための用意ができているかどうかを信号で知らせる（準備できているときは１，準備できていないときは０）Ｓ$_n$（ｎ＝０，１，２，３，．．．，Ｎ－１）を含む第２のレジスタとである。もちろん、ＴＰ$_n$の値はΔＴの倍数である。ディスプレイ用のバッファスイッチングもフレームの境界（すなわち、時間ｔ＝ｍΔＴ，ｍ＝０，１，２，．．．）で生じる。好ましい実施例はＮフレームを含む循環バッファを用いるので、すべてのインデックス（．．．，ｎ－１，ｎ，ｎ＋１，．．．）はモジュロＮインデックスとみなされる。現在のビデオ・フレームを復号化した後の時間がＴであるとする。復号化された現在のフレームは図４７のバッファｎ－１に記憶される。したがって、図４７の次のフレームを記憶するのに用いられるバッファはバッファｎである。

【０２５７】ビットストリームにおける現在の位置を決

53

定する。現在復号化されているフレームのフレーム・インデックスmは次式のように定義される。

【０２５８】

【数６２】

$$m = TP_{n-1} / \Delta T \qquad (2)$$

【０２５９】次のフレームの復号開始時間を決定する。バッファｎにおけるフレームは｛ＴＰｎ以上ＴＰｎ+1以下のｔ｝の時間間隔の間表示されることになるので、バッファｎはＴＰｎ+1まで次のフレームを復号化するのに利用できない。したがって、次のフレームＴｓの復号化開始時間は、次式で表わされる。

【０２６０】

【数６３】

$$Ts = \max\{T,\ TP_{n+1}\} \qquad (3)$$

【０２６１】復号化されるべき次のフレームを決定する。

【外１】

$$\hat{T}d$$

を次のフレームを復号するための見積り時間であるとすると、次のフレームの表示時間は次式を満足しなければならない。

＊

$$TP_n = \max\left\{\Delta T\left[\frac{Ts+\hat{T}d}{\Delta T}+0.5\right], TP_{n-1}+\Delta T\right\} \qquad (4)$$

【０２６７】

【外２】

$$\hat{T}d$$

を見積もるために、先の復号化またはフレーム・パラメータに基づいた統計的な見積りのような異なる方法がある。ある好ましい実施例は、単純に、同じ写真符号化タイプ（Ｉフレームまたは Ｐフレーム）の最も新しく符号化されたフレームの実際の復号化時間＋次のフレームについての見積り復号化時間としてある量の安全マージンを用いる。

【０２６８】こうして、復号されるべき次のフレームのフレーム・インデックスm’は、次式のように計算することができる。

【０２６９】

【数６７】

$$m' = TP_n / \Delta T \qquad (5)$$

【０２７０】続いて、現在の位置からスキップされるべきフレームの数Δｍが、次式によって決定される。

【０２７１】

【数６８】

$$\Delta m = m' - m - 1 \qquad (6)$$

【０２７２】式（２）から式（６）は、環状バッファを更新するための基礎制御動作を構成する。

【０２７３】好ましい実施例は、同期を実現するのに環

54

＊【０２６２】

【数６４】

$$\begin{cases} TP_n > Ts + \hat{T}d \\ TP_n \geq TP_{n-1} + \Delta T \end{cases}$$

【０２６３】上記の条件は、次のフレームの復号化がそれの表示時間の前に終わることを意味し、また、次のフレームは、少なくともビットストリームにある現在のフレームの後のフレームに位置する。ＴＰｎはΔＴの倍数でなければならないので、オーディオに同期できる次のフレームは、次の条件を満たす。

【０２６４】

【数６５】

$$\begin{cases} TP_n = \Delta T\left[\frac{Ts+\hat{T}d}{\Delta T}+0.5\right] \\ TP_n \geq TP_{n-1}+\Delta T \end{cases}$$

【０２６５】ここで、［・］は、切り捨てによる整数部分を示す。

【０２６６】したがって、次のフレームの表示時間は、次式によって決定される。

【数６６】

状バッファ構成を用いる。２つの部分（すなわち、ビデオ復号器バッファスイッチ制御およびディスプレイバッファスイッチ制御）がある。図４８は、ビデオ復号器バッファスイッチ制御のフローチャートを示しており、それは２つの段階（すなわち、初期化および再生）を含む。

【０２７４】初期化：循環バッファの初期化では、N_f（N_fは１以上でN以下）のビデオ・フレームが再生を開始する前に復号される。図４８にダッシュ線の囲みで示されるように、初期化には４つのステップがある。

・ステップ０：すべての表示時間レジスタ｛ＴＰｎ，ｎ＝０，１，２，３，．．．，Ｎ－１｝および状態レジスタ｛Ｓｎ，ｎ＝０，１，２，３，．．．，Ｎ－１｝をゼロに設定し、ビデオ復号器をバッファ０（すなわち、ｎ＝０）に切り替え、ビデオ・ビットストリームの始まりを指し示し（すなわち、ｍ’＝Δｍ＝０）、時間をゼロに設定する（すなわち、ｔ＝０）。

・ステップ１：関連する状態レジスタＳｎを１に設定し、Δｍビデオ・フレームをスキップし、フレームｍ’を復号し、バッファｎに復号されたフレームを記憶する。（ループを通る最初の経路でｎ＝０，ｍ’＝０を呼び戻し、それで最初のフレームが復号されてバッファ０に記憶される。）

・ステップ２：復号化開始時間Ｔｓをｔに設定し、次のバッファ（すなわち、ｎ＋＋）に切り替え、式（４），

（５），（６）に従ってＴＰ$_n$，ｍ’，Δｍを更新する。

・ステップ３：復号フレームの数が前もって設定されたフレームの数Ｎ$_f$に到達したかどうかをチェックする。そうであるならば再生に移り、そうでなければステップ１に戻る。

【０２７５】再生：再生の間に循環バッファを更新するのに、６つのステップが含まれる。

・ステップ０：ディスプレイをバッファ０に切り替え、ディスプレイをイネーブルし、時間を０にリセット（すなわち、ｔ＝Ｔ＝０）し、ビデオ復号器をバッファＮ$_f$（すなわち、ｎ＝Ｎ$_f$）に切り替える。

・ステップ１：全ビデオ・シーケンスが復号されるならば、復号をやめるか、さもなければステップ２に移る。

・ステップ２：式（３），（４），（５），（６）に従ってＴｓ，ＴＰ$_n$，ｍ’およびΔｍを更新する。

・ステップ３：時間がＴｓ（すなわち、Ｔｓはｔ以下）に到達するまで待機し、ステップ４に移る。

・ステップ４：関係する状態レジスタＳ$_n$を０に設定し、Δｍビデオ・フレームをスキップし、フレームｍ’を復号し、復号されたフレームをバッファｎに記憶する。

・ステップ５：フレーム復号化が時間内で終わるならば（すなわち、ｔ＜ＴＰ$_n$）、Ｓ$_n$を１に設定して、復号フレームがディスプレイする準備のできていることを指示し、Ｔをｔに設定し、ビデオ復号器を次のバッファ（すなわち、ｎ＋＋）に切り替える。さもなければ、Ｔをｔに設定し、ＤＴを見積もられた

【外３】

$$\hat{T}_d$$

に加え（すなわち、

【外４】

$$\hat{T}_d$$

＋＝ＤＴ（ＤＴ＝Ｎ$_d$ΔＴ）、次の段階で意図的にもうＮ$_d$（Ｎ$_d$は０以上）だけのフレームをスキップし）、現在のフレーム・インデックスｍをｍ’に設定する。ステップ１に移る。Ｎ$_d$は同期を再開する前に画面がフリーズする時間を制御するためのパラメータであることに留意願いたい。

【０２７６】ユーザーは、環状バッファ・サイズ（Ｎ）と再生用の初期遅延時間（Ｎ$_f$）と画面フリーズ時間（Ｎ$_d$）とを自由に決定できる。明らかに、最小バッファ・サイズは３ビデオ・フレーム（すなわち、Ｎ=３）であり、最も小さな遅延時間は１ビデオ・フレーム（すなわち、Ｎ$_f$＝１）である。しかしながら、ビデオ復号スピードが不充分な場合には、循環バッファ初期化の間にＮ－１フレーム（すなわち、Ｎ$_f$=Ｎ－１）を復号することが強く推奨される結果、ビデオ復号器がオーディオ

実時間再生に追いつくために最大の余地を得ることができる。

【０２７７】ディスプレイバッファスイッチ制御：ディスプレイバッファ制御がビデオ復号器バッファスイッチと並列に実行される。好ましい実施例は、ディスプレイバッファスイッチをビデオ・フレーム境界でチェックする。ｔ＝ｍΔＴ，ｍ＝０，１，２，．．．。ディスプレイがバッファｎ－１におけるビデオ・フレームを現在示しているとすると、現在時間（ＴＰ$_n$以上のｔ）および（Ｓ$_n$＝１）が保たれればかつ保たれさえすれば、それは次のバッファすなわちバッファｎに切り替わる。そうでなければ、それはバッファｎ－１に接続される。ここで、（ｔがＴＰ$_n$以上）かつ（Ｓ$_n$＝０）ならば、それは、復号器が時間内でフレームの復号を終えられなかったことを意味する。この場合、バッファｎにおけるビデオ・フレームは廃棄されており、復号器は、バッファｎを再度更新するために、内輪で選択された次のフレームを復号しており、ディスプレイは、（ｔがＴＰ$_n$以上）および（Ｓ$_n$＝１）が成り立つまで、バッファｎ－１におけるフレームを表示し続ける。要するに、好ましい実施例は、ソフトウェアまたはファームウェアを用いて再生するときオーディオとビデオとの間の同期を実現する一方法を提供する。

【０２７８】可変長復号

可変長復号（ＶＬＤ）は、符号器で可変長符号（ＶＬＣ）を用いて生成される復号ビットストリームに含まれている。図１ｂのアイテム１２６を参照のこと。ＶＬＣのために、符号化ユニットに用いられるビットの数がユニットごとに変わる。したがって、復号器は、それを復号するまで符号ユニットに用いられるビットの数が分からない。これにより、復号器が復号処理の間にビットストリーム・バッファを用いることが不可欠になる。ビデオの符号化では、例えば、符号化されるべきフレームは一組のマクロブロックに分解される（図４９を参照）。最も小さなメモリ要件を考慮するとき、ここでの符号化ユニットは、通常、マクロブロックとして定義される。それは、１６×１６画素輝度領域と、クロマ・フォーマット（４：２：０，４：２：２または４：４：４）に依存する対応クロミナンス領域とからなる。確かに、スライス（フレームにおけるマクロブロックの行）またはフレームそれ自体さえも、充分なメモリがあるならば、符号化ユニットとして取り扱うことができる。

【０２７９】図５０は、好ましい実施例のデジタルスチルカメラ（ＤＳＣ）でのビデオ再生を示す。ＤＳＣ応用では、ビデオ・ビットストリームは、前もって捕獲され、高容量ＳＤＲＡＭに記憶され、また、ビデオ復号器がＤＳＰ上に築かれる。復号器がＳＤＲＡＭに直接アクセスするのは極端に高価であるので、オンチップ・ビットストリーム・バッファがＤＳＰ内部メモリで開放される。ビットストリームは、まず、ＳＤＲＡＭからビット

ストリーム・バッファにＳＤＲＡＭを通ってローディングされ、その後、復号器は、ビットストリーム・バッファにおけるビットストリームを用いてビデオを再構築する。ＣＰＵの介入なしでバックグラウンドにおいて走行できるＤＭＡ（直接メモリ・アクセス）を用いてビットストリーム・ローディングが達成されるため、ビットストリーム・ローディング・オーバーヘッドは、主に、ＤＭＡ転送用のレジスタをセットアップするのに用いられる時間に起因するものとなる。

【０２８０】ビットストリーム・バッファの管理に関しては、２つの基礎的要件がある。第１に、バッファ・サイズは最悪の場合をカバーするのに充分な大きさであること。例えば、ビデオの符号化では、マクロブロックを符号化するための理論上の最大ビット数は２５６ワードであり得る（ここでの１ワードは２バイトとして定義される）。この最悪の場合は非常にまれではあるけれども、安全サイドにあるためには、ビットストリーム・バッファ・サイズは２５６ワードでなければならない。第２に、ビットストリーム・バッファは決してアンダーフローしてはならない。すなわち、バッファ管理は、符号化ユニット用のビットストリームが復号されるときそれが利用可能であることが保証されなければならない。

【０２８１】２番目の要件を満足するために、異なる構成がある。最も簡単なものは、各バッファ・アクセスでビットストリーム・バッファにおける復号位置をチェックするというものである。ビットストリーム・バッファは、復号位置が有効バッファ範囲の外にあるときはいつでも、再充填される。復号化はビットごとの動作であるので、この構成は現実的ではなく、それはバッファをいつ充填するのかを決めるのにオーバーヘッドを費やし過ぎる。現実的な構成は、図５１ａに示されるような線形シフト・バッファ方式である。この方式では、ビットストリーム・バッファは復号器によって左から右へと線形にアクセスされ、また、ユニットを復号した後は、ビットストリームの残りはバッファの始めに前方にシフトされ、その後、バッファは次のユニットを復号する前に「フル」に再充填される。図５１ａにおいて、ＰｓおよびＰｄは、ビットストリーム・バッファにおける現在の復号位置およびビットストリーム最終位置をそれぞれ示す。このバッファ方式は２つの不利益を有する。第１に、バッファ・サイズが復号ユニットのビットの平均数よりも大き過ぎるので、ビットストリーム・シフトに多くの時間が費やされる。例えば、ビデオ復号では、バッファ・サイズは最悪の場合をカバーするのに２５６ワードであるが、平均すると、ユニットは１６ワードしか使っておらず、これは各ユニットについて約２４０ワードのシフトがあることを意味する。第２の不利益は、各ユニットを復号した後にビットストリームのローディングを必要とするというものであり、ＤＭＡ転送を発するのに時間を費やさなければならないので、これには更なるオーバーヘッドが費やされる。

【０２８２】より良いバッファ管理方式は、図５１ｂに示されるようないわゆる擬似循環バッファ方式である。この方式では、復号器はビットストリーム・バッファを循環してアクセスする。これにより、線形バッファ方式に必要とされるビットストリーム・シフトは避けられる。ユニットを復号した後に２つの場合がある。この最初の場合は図５１ｂの左手部分のものであり、ビットストリームの残りがバッファの真中に位置する。この場合には、バッファはビットストリームを２回ローディングすることによって充填され、１回は右端用で、それに続いてもう１回が左端をローディングするためである。（注意：ビットストリーム・ローディングがビットストリームをビットストリーム・バッファに循環して書き込むことができるならば、１回のみのローディングが必要である。しかしながら、いつもこのような場合ではない。）第２の場合は、図５１ｂの右手部分に示されており、そこではバッファの真中のみを充填する必要がある。擬似環状バッファ方式は、ビットストリーム・シフトを避けることができるので線形シフティング・バッファよりもより効率的であるが、各ユニットを復号した後に１つまたは２つのビットストリーム負荷が必要となるという不利益をなおも被る。以下の好ましい実施例のハイブリッド環状二重バッファ方式はこの問題を解決する。

【０２８３】図５２の状態０は、サイズの等しい２つのバッファ（すなわち、左バッファおよび右バッファ）を含むハイブリッド環状二重バッファを示す。各バッファには、バッファ充満（「フル」／「非フル」）を指示するフラッグがある。Ｐｓは、ユニットを復号した後の現在の復号位置を指し示す。バッファ・サイズについては、各バッファは符号化ユニットを復号する最悪の場合をカバーし、これによって、ハイブリッド・バッファ・サイズが線形シフティング・バッファまたは擬似環状バッファの２倍になっている。伝統的な２重バッファとは異なり、ここの２つのバッファは、連続するメモリ割当てを有する。すなわち、左バッファがメモリ・マップにおいて右バッファに直接続けられている。復号器は、環状にハイブリッド・バッファにアクセスする。

【０２８４】好ましい実施例のハイブリッド・バッファは、以下の４つの状態を通って動作する。

・状態０：初期化状態であって、左バッファおよび右バッファの双方が完全にローディングされて「フル」に設定され、Ｐｓはハイブリッド・バッファの開始を指し示す。

・状態１：第１のユニットを復号した後に、左バッファ・フラッグを「非フル」に変更する。

・状態２：ユニットを復号した後に、現在の復号位置Ｐｓが右バッファにあり、かつ、左バッファ・フラッグが「非フル」であるならば、左バッファを完全にローディ

59

ングして、左バッファ・フラッグを「フル」に設定する。また、右バッファ・フラッグが「フル」であるならば、それを「非フル」に変更する。そうでなければ、何の動作もとられない。

・状態３：ユニットを復号した後に、現在の復号位置Ｐｓが左バッファにあり、かつ、右バッファ・フラッグが「非フル」であるならば、右バッファを完全にローディングして、右バッファ・フラッグを「フル」に設定す＊

60

＊る。左バッファ・フラッグが「フル」であるならば、それを「非フル」に変更する。そうでなければ、何の動作もとられない。好ましい実施例のプラットフォーム（例えば、図１ｂ）を例（ここで、データは１６ビット・ユニットにある）として挙げて、以下のデータ種類を定義する。

【０２８５】

【表７】

```
typedef struct bitstream {
      SInt bit_ptr;      /* current bit position (0 ~ 16)
       */
      SInt Ps;           /* current decoding position in bitstream
buffer */
      SInt left_flag     /* left buffer flag "full / not-full"
       */
      SInt right_flag    /* right buffer flag "full / not-full"
       */
      USInt *databuf;    /* bitstream buffer
       */
      Long Addr_SDRAM;   /* bitstream address in SDRAM
*/
} Bitstream;
```

【０２８６】表１に示される擬似符号はハイブリッド環状二重バッファ方式を説明する。バッファ初期化機能（）は復号の開始時に一度だけ呼ばれ、一方、ビットストリーム・バッファ更新機能（）は各符号化ユニットを復号した後に呼ばれ、それは、条件が真になるならば、バッファ・フラッグを自動的に更新し、バッファを※

※再ローディングする。表１において、ＢＵＦＳＩＺＥはハイブリッド環状二重バッファのバッファ・サイズを表す。

【０２８７】

【表８】

61 62

```
Void BufferInitialization(
        Bitstream *stream,        /* pointer of bitstream                    */
        )
{

        /*=========================================================*/
        /* Initialization of the hybrid circular-double buffer     */
        /*=========================================================*/
        LoadBuffer(&stream->databuf[0], stream->Addr_SDRAM, BUFSIZE);
        stream->Addr_SDARM +=BUFSIZE;
        stream->left_flag = "full";
        stream->right_flag = "full"
        stream->Ps  = 0;
        stream->bit_ptr = 16;
}


Void BitstreamBufferUpdate(
        Bitstream *stream,        /* pointer of bitstream                    */
        )
{

        /*=========================================================*/
        /* Update the left buffer if necessary                     */
        /*=========================================================*/
        if (stream->left_flag =="not-full"  && stream->Ps >= BUFSIZE/2)
        {
          LoadBuffer(&stream->databuf[0], stream->Addr_SDRAM, BUFSIZE/2);
          stream->Addr_SDARM +=BUFSIZE/2;
          stream->left_flag = "full";
        }


        /*=========================================================*/
        /* Update the right buffer if necessary                    */
        /*=========================================================*/
        if (stream->right_flag =="not-full"  && stream->Ps < BUFSIZE/2)
        {
          LoadBuffer(&stream->databuf[BUFSIZE/2], stream->Addr_SDRAM, BUFSIZE/2);
          stream->Addr_SDARM +=BUFSIZE/2;
          stream->right_flag = "full";
        }

        /*=========================================================*/
        /* Update the left buffer flag                             */
        /*=========================================================*/
        if (stream->left_flag =="full"  && stream->Ps < BUFSIZE/2)
          stream->left_flag = "not-full";

        /*=========================================================*/
        /* Update the right buffer flag                            */
        /*=========================================================*/
        if (stream->right_flag =="full"  && stream->Ps >= BUFSIZE/2)
          stream->right_flag = "not-full";

}
```

表１．ハイブリッド環状二重バッファ構成のための擬似符号

【０２８８】表１におけるBitstreamBufferUpdate()に見ることができるように、左バッファまたは右バッファは、各ユニットを復号した後に再ローディングされないが、反対バッファ（左／右）が用いられていて、かつ、それのバッファ・フラッグが「非フル」でありさえすれば、ローディングされる。これはバッファ負荷の数を大きく低減する。ビデオ符号化を一例として考える。これは、マクロブロックがユニットであって、ユニットの平均ビットストリーム・サイズが１６ワードであるとするならば、５１２ワードのBUFSIZEを必要とする。線形シフティング・バッファおよび擬似環状バッファが各ユニットを復号した後にバッファを再充填するので、これらの２つの方式の平均ローディング長も１６ワードである。ハイブリッド環状二重バッファ方式における２５６ワードの固定ローディング長と比較して、好ましい実施例は約１６のファクターでローディング・オーバーヘッドを低減する（すなわち、２５６／１６）。

【０２８９】ミニ実験をして、上述した３つのバッファ方式を比較した。使われたビデオ･シーケンスは、コーストガード（３５２×２８８，３００フレーム、４：２：０）であった。ビットストリームはＭＰＥＧ１ビデオ符号器を用いて生成されている。目標ビット速度は３メガビット／秒，Ｉフレームのみである。３つの異なるバッファ方式の同じ復号器が同じビットストリームを復号するために用いられ、バッファ・ローディング・カウントおよびワード・シフティング・カウントが復号化の間に記録される。３つのバッファ方式の間の性能の比較が表２に列挙されている。表２に示されるとおり、各マクロ・ブロックについて、線形シフティング・バッファ方式は、１つのバッファ負荷と、平均で約２４０ワードのシフティングとを必要とする。擬似環状バッファ方式は、わずかに多くのバッファ負荷を必要とするが（１．０６負荷／マクロブロック）、シフティングを必要としない。好ましい実施例のハイブリッド環状二重バッファ方式は、マクロブロック当り約０．０６１９バッファ負荷だけを用いた。特に、図１ｂの好ましい実施例のプラ

ットフォームでは、好ましい実施例の方式は、線形シフティング・バッファ方式および擬似環状バッファ方式と比較して約１１３および１７のサイクル・カウント低減＊比をそれぞれ提供する。

【０２９０】

【表９】

	線形シフティング・バッファ	擬似環状バッファ	ハイブリッド環状二重バッファ
バッファ・サイズ（ワード）	２５６	２５６	５１２
マクロブロック当りの負荷の数	１．００	１．０６	０．０６１９
マクロブロック当りのワード・シフティングの数	２４０．１５	０	０
負荷（サイクル）当りのオーバーヘッド	８０	８０	８０
ワード・シフティング当りのサイクル・カウント	２	２	２
マクロブロック当りのビットストリーム・バッファに用いられる合計サイクル	５６０．３０	８４．７２	４．９５
サイクル・カウント比対ハイブリッド環状二重バッファ方式	*113.19*	*17.12*	*1.00*

表２．TMS320DSC21プラットフォーム上の３つのバッファ方式間の性能比較

【０２９１】画面上表示およびグラフィック加速

画面上表示（ＯＳＤ）モジュール１０５は、異なるＯＳＤウインドウからのＯＳＤデータを管理する責任があり、また、それをビデオと混合する。それは、ＯＳＤデータをＳＤＲＡＭ１６０から読み出し、ＮＴＳＣ／ＰＡＬ符号器１０６に出力する。ＯＳＤモジュールはスタンバイ・モジュールにデフォルトし、そこでは、それはビデオをＮＴＳＣ／ＰＡＬ符号器１０６に単純に送る。ＡＲＭ ＣＰＵ１３０によって構成され活性化された後に、ＯＳＤモジュールはＯＳＤデータを読み出して、それをビデオ出力と混合する。ＡＲＭ ＣＰＵ１３０は、ＯＳＤ動作をオン／オフするとともにＯＳＤデータをＳＤＲＡＭに書き込む責任がある。図１５は、ＯＳＤモジュールのブロック図および関係する他のアイテムを示す。ＯＳＤの様々な機能が、以下の段落で説明される。

【０２９２】ＯＳＤデータ記憶。ＯＳＤデータは様々なサイズを有する。ビットマップ・ウインドウでは、各画素は１，２，４または８ビット幅であり得る。ＹＣｒＣｂ４：２：２ウインドウでは、それは成分当り８ビットを取り、その成分は４：２：２（Ｃｂ／Ｙ／Ｃｒ／Ｙ．．．）フォーマットに従って配列される。ＲＧＢグラフィック・データがＯＳＤとして用いられる必要がある場合には、アプリケーションは、それを記憶する前にＹ／Ｃｒ／Ｃｂへのソフトウェア変換を行う。ＯＳＤデータは、常に、３２ビット・ワードにパックされ、正しいままとされる。ＯＳＤウインドウの左上角から始めて、すべてのデータは隣接３２ビット・ワードにパックされる。

【０２９３】ＯＳＤウインドウを設定する。ＯＳＤウインドウはそれの属性によって定義される。ウインドウ用のＯＳＤデータをＡＲＭ ＣＰＵ１３０によってＳＤＲ※ＡＭに記憶することに加えて、アプリケーション・プログラムも、以下のサブセクションで説明される通り、ウインドウ属性およびＯＳＤモジュールにおける他の設定を更新する必要がある。

【０２９４】位置レジスタ。位置レジスタは各ウインドウの左上角および右下角のＸおよびＹの位置を含む。アプリケーション・プログラムは、ＣＡＭを設定し、選択されたＯＳＤウインドウをイネーブルする必要がある。図１６を参照。

【０２９５】カラー・ルックアップ・テーブル。ＯＳＤは固定２５６エントリ・カラー・ルックアップ・テーブル（ＣＬＵＴ）を有する。ＣＬＵＴはビットマップ・データをＹ／Ｃｒ／Ｃｂ成分に変換するのに用いられる。１，２または４ビットマップ画素の場合には、ＣＬＵＴはＣＬＵＴレジスタによって決定することができる。

【０２９６】混合および透明性。画素レベルでのカラー混合がまた支援される。この特徴はビットマップ表示のためだけに利用できる（ウインドウ１，２）。ウインドウ・カラー混合がイネーブルされるならば、各画素の混合量は混合ファクターによって決定される。以下の表に示される通り、ウインドウ混合は、選択された混合ファクターに従って５つの異なるレベルを支援する。ハードウェアはビットマップで透明性モードを支援する。透明性がイネーブルされるならば、０の値を有するビットマップ表示上のいかなる画素も、ビデオが表示されることを可能とする。本質的に、０の値の画素は、透明なカラー（すなわち、背景カラーがビットマップを通して示される）と考えられる。表には、同じウインドウ上の透明性と混合との間の接続が示されている。

【０２９７】

【表１０】

65　　66

透明性	混合ファクター	ＯＳＤウインドウ寄与度	ビデオ寄与度
オフ	0	0	1
	1	1／4	3／4
	2	1／2	1／2
	3	3／4	1／4
	4	1	0
オン		画素値＝０ならば、	画素値＝０ならば、
	0	0	1
	1	1／4	3／4
	2	1／2	1／2
	3	3／4	1／4
	4	1	0

【０２９８】ハードウェア・カーソル。長方形がハードウェア・ウインドウ１を用いて提供される。ウインドウ１では、カーソルは，常に，他のＯＳＤウインドウ上に現れる。ユーザは、その形のサイズおよびカラーを特定できる。ハードウェア・ウインドウ１がカーソルとして指定されるとき、２つのウインドウのみがＯＳＤアプリケーションに利用できる。ハードウェア・カーソルが用いられないならば、アプリケーションはウインドウ１を通常のハードウェア・ウインドウとして用いることができる。図１７は、ハードウェア・カーソルの一例を示す。

【０２９９】ＤＳＰサブシステム

ＤＳＰサブシステムは、Ｃ５４ｘＤＳＰと、ローカルメモリ・ブロックと、ｉＭＸおよびＶＬＣ加速器と、共有される画像バッファと、共有を実行する乗算器とからなる。Ｃ５４ｘは、高性能，低電力および市場で証明されたＤＳＰである。Ｃ５４ｘ用のｃＤＳＰハードウェアおよびソフトウェア開発ツールも非常に円熟している。ＤＳＰは、オート露光，オートフォーカスおよびオートホワイトバランシング（ＡＥ／ＡＦ／ＡＷＢ）と画像パイプライン・タスクの一部とを実行する。それはまた、ＳＤＲＡＭ転送を取り扱い、加速器を駆動して画像処理の残りおよび画像圧縮タスクを実行する。ＤＳＰにおけるプログラミングの柔軟性および簡易性により、カメラ製造業者が、画像処理フローを洗練させ、品質および性能のトレードオフを調整し、カメラに更なる特徴を導入することが可能となる。

【０３００】適合性のあるＤＳＰ（ｃＤＳＰ）設計フローは、柔軟性および設計再使用を可能とするように採用される。ＤＳＰと加速器との間でタイム・シェアリングされるメモリ・ブロックは、１つの処理ユニット（１６×１６画素）のために十分大きく、また、ＤＳＰへのゼロ待機状態アクセスを提供する。

特徴

固定小数点デジタル信号プロセッサ

１００ＭＩＰｓＬＥＡＤ２．０ＣＰＵ

オンモジュールＲＡＭ３２Ｋ×１６ビット

（４ブロックの８Ｋ×１６ビット二重アクセス・プログラム／データＲＡＭ）

マルチチャンネルバッファされたシリアルポート（ＭｃＢＳＰｓ）

ＡＲＭは拡張８ビット・ホスト・ポート・インターフェースを介してＲＡＭにアクセスすることができる。

１つのハードウェア・タイマー

オンチップ・プログラム可能ＰＬＬ

ソフトウェア・プログラム可能待機状態生成器

スキャン・ベース・エミュレーションおよびＪＴＡＧ境界スキャン・ロジック

【０３０１】図１８ａは、ＤＳＰサブシステムについての更なる詳細、特に、ＤＳＰとｉＭＸおよびＶＬＣとの間の接続の詳細を示す。図１８ｂは、メモリ・マップである。共有されたメモリ・ブロックＡ，Ｂは、ＤＳＰのデータ・メモリ空間上で２つの２Ｋワード・バンクを占める。各ブロックは、ＤＳＰによって制御される状態スイッチングに依存してＤＳＰ，ｉＭＸ，ＶＣＬおよびＳＤＲＡＭコントローラによってアクセスすることができる。ダイナミックなサイクルごとのメモリ仲裁は計画されない。ＤＳＰのプログラムは、ゼロ待機状態外部メモリ・インターフェースを通してこれらのメモリ・ブロックの継ぎ目のないアクセスを得る。ｉＭＸ係数，ｉＭＸコマンド，ＶＬＣ Ｑ行列およびＶＬＣハフマン表用の構成メモリ・ブロックもＤＳＰの外部メモリ・インターフェースに接続する。それらはまた、特定のモジュールとＤＳＰとの間で静的に切り替えられる。典型的には、パワーアップでまたはカメラ動作モードの初期の段階で、これらのメモリ・ブロックはＤＳＰ側に切り替えられ、ＤＳＰが動作のために適切な構成情報を設定することができるものとなっている。その後、それらは、動作の期間中にｉＭＸおよびＶＬＣに切り替えられる。

【０３０２】イメージング拡張（ｉＭＸ）

ｉＭＸ（イメージング拡張）は、プログラム可能なＤＳＰの画像処理性能を拡張するための柔軟な制御およびメモリ・インターフェースをもつ並列ＭＡＣエンジンである。ｉＭＸは、柔軟性，メモリ使用およびプログラミングの簡易性が達成されるように、ＤＳＰプロセッサを備えた共有メモリ構成において良好に作動するものと考えられる。そのアーキテクチャは、一般の１－Ｄおよび２－Ｄ ＦＩＲフィルタリング，アレイ・スケーリング／付加，（カラー空間変換用の）行列乗算，クリッピングおよび閾値動作をカバーする。デジタルスチルカメラでは、ｉＭＸは、以下のものをスピードアップするのに用いることができる。

ＣＦＡ補間

カラー空間変換

クロマ・ダウンサンプリング
エッジ強調
カラー抑制
ＤＣＴおよびＩＤＣＴ
テーブル・ルックアップ

【０３０３】ｉＭＸ方法論は、並列処理および高性能コンピュータ・アーキテクチャの研究から生じている。設計は、スケーラブルＭＡＣエンジンの必要性を含む。第１の好ましい実施例におけるｉＭＸは４つのＭＡＣユニットを組み込んでいる。図１９を参照。代替の好ましい実施例は８以上のＭＡＣユニットにグレードアップしている。ソフトウェアは、ハードウェア・グレードアップが実質的なソフトウェア変更を生じないように、構成することができる。ｉＭＸの大きな柔軟性は、パラメータ起動アドレス生成およびルーピング制御によるものである。全体の効率は、ｉＭＸ内の効率的なパイプライン制御およびシステム・レベル・メモリ・バッファリング方式から生じている。ｉＭＸはブロック・ベース処理のために最もよく作動する。これを容易とするために、データパスはデータ入出力および係数メモリに接続する必要がある。ｉＭＸは、データ入力ポート，データ出力ポートおよび係数メモリ・ポートを含み、また、これらのポート間の仲裁を可能とする。これは、専用メモリ・ブロックの必要性を削除し、システムレベルに更なる柔軟性とより良いメモリ使用とをもたらす。これらのメモリ・ブロックは、データ交換を容易とするために、ＤＳＰデータ・メモリとしてアクセス可能である。ｉＭＸにコマンド復号ユニットを送出する別のコマンド・メモリがある。そのコマンド・メモリが、自らの基準画像パイプライン・アルゴリズムにおいて、すべての加速されたステップに適合するよう指定され、このコマンドのシーケンスがＤＳＰからの介入をほとんどなしに実行することができるものとされる。

【０３０４】ｉＭＸブロック図が図２０に示されている。コマンド復号サブブロックは、コマンドを読み出し復号し、また、静的パラメータをコマンド当り一組だけアドレス生成器に押しやる。その後、アドレス生成器は、ルーピング変数およびデータ／係数／出力ポインタを計算し、サイクルごとのパイプライン制御を取り扱う実行制御と整合する。アドレス生成器は、データおよび係数読出し要求をアービタ（arbiter）に送る。アービタはその要求をデータ／係数メモリに転送する。メモリから読み戻されたデータは入力フォーマッタに行き、それはデータの整列および複製を行う。その後、フォーマットされたデータおよび係数はデータパスに提供され、それは、主として、４つのＭＡＣユニットからなる。データパスからの出力は、メモリ書込み用のアービタへの経路をたどる。ｉＭＸは、共有メモリを介して（データ入力，係数，データ出力，コマンドのために）およびメモリ・マップされたレジスタ（開始コマンド，終了状態）を介してＤＳＰと通信する。すべてのデータ・バッファおよびメモリ・ブロックは、シングル・ポートであり、オンライン仲裁よりもむしろ静的制御を介して１つのものから別のものへと切り替えられる。

【０３０５】典型的な応用では、ＤＳＰは、フィルタ係数，ＤＣＴ／ＩＤＣＴコサイン定数およびルックアップ・テーブルを係数メモリに置き、また、ｉＭＸコマンドをコマンド・メモリに置く。その後、ＤＳＰはこれらのメモリ・ブロックへのアクセスをｉＭＸに引き渡す。これらのメモリ・ブロックは、主要なカメラ動作モード（例えば、画像捕獲）のために必要なすべての係数およびコマンドに適合するために、基準設計に適切なサイズとされる。いかなる更新／再ローディングも非常に低い頻度で起こるべきである。いずれかのまたは双方のメモリ・ブロックの空間がなくなる場合には、ページングを行うことができる。ＤＳＰは、ｉＭＸに１つのデータ・バッファだけがあるように、スイッチ・ネットワークを管理する。走行時間の間、ＤＳＰは、Ａ／Ｂバッファをそれ自体，ｉＭＸ，ＶＬＸおよびＳＤＲＡＭコントローラの間で切り替えて、データの受け渡しを実行した。図２１は、ＤＳＰ上の画像パイプラインをスピードアップするのに用いられる入力切上げ／クリッピング能力を備えた単純テーブル・ルックアップ加速器を図示する。これは非常に単純な制御構造およびデータパスで実行される。

【０３０６】ＶＬＣエンジン

ＶＬＣ加速器は、ＪＰＥＧ圧縮およびＭＰＥＧ圧縮の場面における量子化およびハフマン符号化のために最適化されたコプロセッサである。それは、ＤＳＰによってプレローディングされた量子化行列およびハフマン表で共有メモリ・ブロックを介して動作する。設計における積極的なパイプラインは、圧縮用の３千万のＤＣＴ係数を超える非常に高いスループット速度を達成する。量子化行列，ハフマン表およびデータ入出力メモリを含むＶＬＣの作動メモリはすべて共有メモリ・ブロックである。

【０３０７】ＶＬＣ機能性

基本的に、ＶＬＣは、量子化，ジグザグ・スキャンおよびＪＰＥＧ符号用のハフマン符号（ベースラインＤＣＴ，８ビット・サンプル）を４つまでの量子化行列（invq[i，j]＝2^{16}／q[i，j]）およびすべてローディング可能な２つの符号化ハフマン表でカバーする。１０ブロックまでを含む１つのＭＣＵを処理することができる。各ブロックは８×８＝６４サンプルからなる。量子化，ジグザグ・スキャンおよびＭＰＥＧ－１ビデオ符号化用のハフマン符号化。６つまでの８×８ブロックで１つのマクロブロックを処理することができる。ブロックの数およびそれらの中の輝度ブロック数を指定することができる。ハフマン符号化は、量子化されジグザグに順位付けられたレベルを生成するのに迂回することができる。加速器は、入出力バッファ，量子化行列およびハフマン

符号化表用のメモリ・ブロックを要求する。メモリ構成は、通常の符号化動作，１つのＪＰＥＧ ＭＣＵ（最小符号化ユニット）または呼出し当りのＭＰＥＧマクロブロックを支援するのに十分なものである。

【０３０８】入力および出力の双方は２Ｋワード（１ワード＝１６ビット）共有メモリ・バッファ（ＡまたはＢ）に適合しなければならない。ＭＣＵまたはマクロブロックは、最大１０の８×８ブロックまたは６４０入力ワードを有する。圧縮された出力データは、典型的には、入力サイズよりも小さい。ＪＰＥＧハフマン符号化表は、表当り（１２×１７６）×３２ビットまたは３８４ワードを費やす。ＪＰＥＧ標準は、２つの表を可能とし、合計で７６８メモリ・ワードを費やす。ＭＰＥＧ表は、ＶＬＣに回路接続され、メモリを費やさない。ハフマン表には２Ｋワードを割り当てている。１６ビットによる５１２ワードの量子化行列メモリは、各々が６４×１６ビットを費やす８つの量子化行列が同時に存在することを可能とする。ＪＰＥＧは、４つの行列を可能とし、ＭＰＥＧ符号化は２つの行列を必要とする。図２２はＶＬＣの主なサブブロックを示す。符号化経路のみが１つの好ましい実施例のＶＬＣモジュールで実現され、代替の好ましい実施例では、復号化経路はモジュールに組み込まれる。

【０３０９】ＡＲＭサブシステム

ＡＲＭマイクロプロセッサ１３０は、システムレベル初期化，構成，ユーザ・インターフェース，ユーザコマンド実行，接続性機能および全体のシステム制御を取り扱う。ＡＲＭ１３０は、大きなメモリ空間，より良いコンテクスト切替え能力を有しており、それにより、ＤＳＰ１２２よりも複雑な処理，マルチ・タスキング処理および一般の処理に適当である。好ましい実施例は、ＡＲＭ７ｃＴＤＭＩコアを一体化する。図１ｂを参照。ＡＲＭ７コアは少なくとも４０ＭＨｚまで指定される。ＡＲＭサブシステムはまた、３２Ｋバイト・ローカル・スタティックＲＡＭ１３２を有する。ＡＲＭプロセッサ１３０は、ＣＣＤコントローラ，ＴＶ符号器，プレビュー・エンジン，ＩｒＤＡ，ＵＳＢ，コンパクトフラッシュ（登録商標）／スマートメディア，ＵＡＲＴなどを含むすべてのＤＳＣ周辺機器に接続される。

【０３１０】ＡＲＭプロセッサ１３０は、ＳＤＲＡＭおよびＬＣＤＮへのＣＣＤ入力生データおよび中間データの管理をする。すべてのＩ／Ｏデバイスに接続されて、ＡＲＭは、ＵＳＢ，ＩｒＤＡ，コンパクトフラッシュ／スマートメディアおよびＵＡＲＴＳのようなスマートデバイスを管理し、それらに対して責任がある。ＰＲＥＶＩＥＷ，ＣＡＰＴＵＲＥ，ＰＬＡＹＢＡＣＫおよびＢＵＲＳＴの４つの基礎動作モードはＡＲＭからの要求によって開始される。その後、ＡＲＭは、その要求を完了するためにデバイスをモニタし、場合によっては、その要求が完了した後にデータを管理する。ＲＥＳＥＴの後で

かついかなるカメラ動作が起こり得る前に、ＡＲＭはいくつかのハウスキーピング・タスクを行わなければならない。最初のタスクはＢＯＯＴ動作タスクとして知られている。この機能はＩ／Ｏおよび周辺機器を周知の状態に初期化するだけではなく、それはまたＤＳＰ１２２を用意しローディングし開始しなければならない。このシーケンスは、フラッシュからＤＳＰブート符号を読み出し、ＤＳＰ符号メモリをローディングし、続いてそれのＨＯＬＤ状態からＤＳＰを放出することによって、開始する。更なるＤＳＰ符号があるフォーマットでＳＤＲＡＭにロードされ、その後、ＤＳＰは、ＡＲＭ介入なしにそれの符号空間を読み出しオーバーレイすることができる。

【０３１１】ＡＲＭ ＳＤＲＡＭインターフェース

ＡＲＭは、（１）ＳＤＲＡＭバッファ（バースト読出し／書込み）を通して、（２）より長い待機時間－４サイクルＲＥＡＤ，６サイクルＷＲＩＴＥでのＳＤＲＡＭへの直接アクセスという、ＳＤＲＡＭへの２種類のアクセスを有する。メモリへの直接アクセスはワード，半ワードまたはバイト・アクセスであり得る。ＡＲＭ／ＳＤＲＡＭコントローラ・インターフェースはまた、３２バイトのバッファを有する。ＳＤＲＡＭバースト要求は最初にこのバッファを充填し、ＡＲＭはこのバッファからの読出しとそれへの書込みとを行う。

【０３１２】ＡＲＭ外部メモリ・インターフェース

ＡＲＭ１３０は、外部メモリ・インターフェース・モジュールを通して外部メモリに接続する。ＡＲＭ１３０は、このインターフェースを通してコンパクトフラッシュ／スマートメディアに接続する。ＡＲＭ１３０はまた、このインターフェースを通してオフチップのフラッシュメモリに接続する。ＤＭＡブロック（図１ｂ）は、ＡＲＭをＣＦ／スマートメディア転送に高める。

【０３１３】ＡＲＭ／ＤＳＰ ＢＯＯＴシーケンス

ＤＳＰ ＢＯＯＴシーケンスは電源立上げの後かＣＯＬＤ ＳＴＡＲＴの後に開始する。この状態で、ＤＳＰ１２２は、ＡＲＭ１３０からの初期化を待機するＨＯＬＤ状態にある。ＡＲＭは、ＤＳＰ状態レジスタをチェックして、ＤＳＰがＨＯＬＤ状態にあることを確認する。ＡＲＭは、ＤＳＰブート符号データをＦＬＡＳＨからのＤＳＰ符号メモリにプログラムする。その符号は、必要とされる機能のために適当な符号（この場合には、ＢＯＯＴ符号）をＡＲＭが選択するのを可能とする論理オーバーレイにおいて整理される。ＡＲＭは、ＨＰＩブリッジ（ＨＰＩＢ）インターフェースを用いてＤＳＰ符号をローディングする。このインターフェースは、８ビット幅か１６ビット幅でアクセスするようにプログラムできる。ＢＯＯＴの目的のために、これは常に１６ビット・アクセスである。符号がロードされた後に、ＡＲＭは、ＤＳＰに信号を送ってＨＯＬＤを開放することによって開始させる。その後、ＤＳＰは、ＤＳＰ ＲＥＳＥＴベ

クトル領域にあるＤＳＰ ７Ｆ８０ｈのアドレスからそれのリセット・シーケンスを開始する。ＲＥＳＥＴシーケンスを完了すると、ＤＳＰは、ＡＲＭによってロードされたＢＯＯＴプログラムの開始であるＤＳＰ ＦＦ８０ｈに分岐する。図２３ａは、ＡＲＭ／ＤＳＰブート・シーケンスで用いられるデータパスと、後述されるデータ，要求およびコマンド交換とを示す。

【０３１４】捕獲モード

ＡＲＭ１３０は、画像を捕獲するようにＣＣＤコントローラ１０２をプログラムする。ＣＣＤコントローラは、画像データをＳＤＲＡＭに自動的に転送し、その転送が完了したときにＩＲＱ１を用いてＡＲＭを中断する。その後、ＡＲＭは、生映像データがクランチするのに利用可能であるということをＤＳＰに通知する。生データの処理が完了したとき、ＤＳＰは、タスクが終了されたという信号をＡＲＭに送る。

【０３１５】プレビュー・モード

ＣＣＤは、３０ｆｐｓの高フレーム速度用にプログラムされているが、垂直の解像度が低減されている。ＣＣＤおよびＴＧ（タイミング生成器）の再構成は、生映像データをプレビュー・エンジン１０４に行かせる。ＤＳＰは、ＳＤＲＡＭにおいてデータを後処理し、ＦＯＣＵＳ，ＥＸＰＯＳＵＲＥおよびＷＨＩＴＥＢＡＬＡＮＣＥ用のパラメータを用意する。ＡＲＭは、新しい調整パラメータが準備できるとともにこれらの補正がＡＲＭによって適用されるときに、ＤＳＰによって信号で知らされる。補正パラメータの転送は、前述したのと同じ通信中断アーキテクチャを用い、また、現在のフレーム速度であることが期待される。

【０３１６】バースト・モード

バースト・モード・タイミングは、アプリケーション・パラメータからの映像速度をクロックするＡＲＭに基づく。捕獲モードとプレビュー・モードとの間の交差と同様に、ＡＲＭは、圧縮エンジンを通してＳＤＲＡＭに圧縮画像を記憶する捕獲のためにＣＣＤをプログラムする。プレビュー・モードと同様に、ＡＲＭは、ＤＳＰから調整パラメータを受け取ってＦＯＣＵＳ，ＥＸＰＯＳＵＲＥおよびＷＨＩＴＥ ＢＡＬＡＮＣＥの補正を行う。

【０３１７】アイドル・モード

ＡＲＭは、アイドル・モードを用いて、他のカメラ・モードに先行する間にＤＳＰから補正パラメータを受け取る。電源ダウン状態になければ、１０～１５フレームのこの時間は、ＤＳＰからＡＲＭへの補正ループがＦＯＣＵＳ，ＥＸＰＯＳＵＲＥおよびＷＨＩＴＥ ＢＡＬＡＮＣＥについて自動補正を行うことを可能とする。このアイドル・モードは、安定した補正を得るという目的でプレビュー・モードをシミュレーションする。

【０３１８】ＡＲＭ／ＤＳＰコミュニケーション

ＡＲＭ１３０とＤＳＰ１２２との間の通信は、ＨＰＩＢ（ホスト・ポート・インターフェース・ブリッジ）を介する。ＨＰＩＢは、ＤＳＰ（Ｃ５４０９タイプのＤＳＰ）ポートとＢＵＳＣ（ＢＵＳコントローラ）１３４とを物理的に接続する。ＡＲＭは、ＨＰＩＢをプログラムするとともにＤＳＰメモリ・マップに３２ｋワード・ウインドウを開けることによって、ＤＳＰメモリにアクセスする。マップは、コマンド要求，アクノリッジおよびデータグラム（datagram）のためにＡＲＭおよびＤＳＰによって共有されるデータ構造を含む。ＨＰＩＢは５つのサブブロックを含む。それらは、インターフェースと、タイミング生成器と、ＤＳＰ制御レジスタと、中断保持セクションとである。インターフェース・セクションは、ＢＵＳＣ１３４からのデータを受信して記憶し、それをＣ５４０９におよびそこから転送する。このインターフェースは、Ｃ５４０９への８ビットまたは１６ビット・データパスであり得、また、ＢＵＳＣへの１６ビットである。付加された特徴は、そのようにプログラムされるならば、上部バイトおよび下部バイトを交換する能力である。タイミング生成器は、信号ＨＢＩＬおよび信号ＨＤＳを作り、信号ＨＲＤＹを検出する。ＨＢＩＬはＣ５４０９へのＨＰＩバイト識別信号である。ＨＤＳはＣ５４０９へのデータ・ストローブ信号であり、ＨＲＤＹはＣ５４０９から読み出されたレディ信号である。中断保持セクションは、ＨＩＮＴレベルを検出し、ＩＮＴＣパルスをＡＲＭクロックと同期させる。モジュールはまた、Ｃ５４０９のＨＯＬＤポートを設定し、ＨＯＬＤＡを検出する。

【０３１９】８ビット・モードでは、ＡＲＭからのアドレス・データはＣ５４０９に到達しない。アドレスは、Ｃ５４０９内部メモリが選択されていさえすれば、用いられる。したがって、ＡＲＭは、３２ＫワードＤＡＲＡＭにデータを送るか受け取るかする前にＨＰＩＡレジスタにアドレスを設定しなければならない。８ビット・モードはまた、ＡＲＭとＤＳＰとの間のハンドシェーキングのために用いられる。ＡＲＭは、ＨＰＩＣレジスタにおけるＨＩＮＴビットを用いてＣ５４０９を中断する。１６ビット・モードでは、ＨＰＩＡ／ＨＰＩＣ／ＨＰＩＤは用いられない。ＡＲＭは、あたかもそれがＨＰＩＢモジュールに存在するかのように、Ｃ５４０９内部メモリにアクセスできる。このモードは、より速いパフォーマンスを行うが、ＨＡＮＤＳＨＡＫＥ信号はＨＰＩＣレジスタに経路付けられているのでＨＡＮＤＳＨＡＫＥ信号を支援しない。

【０３２０】図２３ｂは、ＡＲＭがＣ５４０９ＤＡＲＡＭに到達する信号および経路を示す。図２３ｃは、ＡＲＭ（ＨＯＳＴ）とＣ５４０９プロセッサとの間の共有メモリ・マップを示す。ＡＲＭがメモリ領域「ＤＳＰメモリ」を選択するとき、ＢＵＳＣは、ｃｓ＿ｈｐｉｂ信号をアクティブにする。ここで、ＡＲＭはＤＳＰ内部メモリ（３２ｋワードＤＡＲＡＭ＋ＨＰＩＡ＋ＨＰＩＣ＋Ｈ

ＰＩＤ）にアクセスすることができる。ＡＲＭが、「ＤＳＰコントローラ」領域を選択するとき、ＢＵＳＣはｃｓ＿ｄｓｐｃ信号をアクティブにする。ここで、ＡＲＭは、Ｃ５４０９に関係するレジスタにアクセスしている。

【０３２１】多重処理デバギング環境

好ましい実施例は、ＡＲＭ１３０とＤＳＰ１２２したがって多重処理を一体化し、それにより、デバギングおよび開発支援を必要とする。好ましい実施例は、図２４に例示される付加的なエミュレーション論理を備えた一つのＪＴＡＧコネクタ１７０でこれを達成する。

【０３２２】入力／出力モジュール

入力／出力モジュールは、以下のようにＤＳＣ周辺機器に異なるインターフェースを提供する。ＴＶ符号器１０６は、ＬＣＤディスプレイおよびＴＶ用のＮＴＳＣ／ＰＡＬおよびＲＧＢ出力を生成する。ＣＣＤ／ＣＭＯＳコントローラ１０２は、タイミング信号ＶＤ／ＨＤを生成し、外部で生成されたＨＤ／ＶＤ信号（ＭＯＤＥＳＥＴレジスタの＃０、ＳＹＮＣＥＮレジスタの＃０）について同期し、プログレッシブ・スキャンおよびインターレースＣＣＤを支援し、ブラック・クランピング制御信号，プログラム可能なｃｕｌｉｎｇパターン（９ＣＵＬＨ，ＣＵＬＶレジスタ），１ライン／２ライン交代フィールド，（ＣＣＤモジュールによって生成される）ＭＣＬＫを生成し、ＷＥＮ（ＴＧ上のＷＲＱ、アクティブ－ハイ）は、ＳＤＲＡＭにデータを書き込むＣＣＤコントローラを指示し、ＴＧシリアル・ポート・インターフェースはＧＩＯピンによって制御され、アイリス，機械シャッター，焦点およびズームはＧＩＯピンによって制御される。

【０３２３】プログラマーの視野からのＵＳＢ１４２は、３つの主要部分（ＦＩＦＯコントローラ，ＵＤＣコントローラおよびＵＤＣコア）からなる。ＵＳＢ構成：ＩＮＴＥＲＦＡＣＥＤ０ ＡＬＴ０ ＥＮＤＰＯＩＮＴ０： ＣＯＮＴＲＯＬ； ＩＮＴＥＲＦＡＣＥ０ ＡＬＴ０ ＥＮＤＰＯＩＮＴ１： ＢＵＬＫＩＮ； ＩＮＴＥＲＦＡＣＥ０ ＡＬＴ０ ＥＮＤＰＯＩＮＴ１： ＢＵＬＫＯＵＴ； ＩＮＴＥＲＦＡＣＥ１ ＡＬＴ０ ＥＮＤＰＯＩＮＴ２： ＩＳＯＩＮ； ＩＮＴＥＲＦＡＣＥ２ ＡＬＴ０ ＥＮＤＰＯＩＮＴ３： ＩＮＴＥＲＲＵＰＴ ＩＮ。バッファ構成：ＳＵＢモジュールは６つのＦＩＦＯを内部に有する。各ＦＩＦＯは、方向およびバッファ・サイズを除いて、同じ構造である。ＵＳＢモジュールは、すべての終点についてひとつだけの統一メモリを有する。バッファ・サイズは、すべてのバッファがメモリ内部に設けられている限り、プログラム可能である。

【０３２４】Ｉ／Ｏブロック１４０のＵＡＲＴ部は、スタート／ストップ通信プロトコルを支援し、（偶数，奇数パリティまたはパリティなしで７または８ビットのおよび１または２ストップ・ビットのデータ長を支援する）パリティ・エラーを検出し、送信機および受信機の双方用のＦＩＦＯの３２バイトを有し、ＦＩＦＯオーバーフロー用の中断を生成するかタイムアウトがデータ受信で検出される。ＡＲＭ１３０はＵＡＲＴモジュールを制御する。ＡＲＭ１３０からアクセス可能な７つの１６ビット幅レジスタがある。データ送信機／受信機レジスタ（ＦＩＦＯ）と、ビット速度レジスタと、モード・レジスタと、受信機用ＦＩＦＯ制御レジスタと、送信機用ＦＩＦＯ制御レジスタと、ライン制御レジスタと、状態レジスタとである。図２５はブロック図である。

【０３２５】コンパクトフラッシュ／スマートメディア・インターフェース１８０は、画像またはユーザのデータをコンパクトフラッシュカードまたはスマートメディアに保存／記憶するのに用いられる。図２６参照。インターフェースは、レジスタ設定およびデータ転送用の２種類の動作モードを支援する。すなわち、メモリ・マップド・モードとＩ／Ｏモードとである。コンパクトフラッシュカードがプラグに差し込まれつつあるかプラグから抜かれつつあるかする間、ＡＲＭ１３０中断がカード検出のために生成される。スマートメディアおよびコンパクトフラッシュ制御インターフェースの双方用のピンは、オーバーラップされ、製品ニーズに依存してＡＲＭ１３０によって切り替えることができる。図２６を参照。特に、コンパクトフラッシュコントローラは、ＡＲＭメモリ空間にマップされたレジスタを有する。コンパクトフラッシュコントローラは、インターフェースピンに関連制御信号を生成する責務があり、４２０ＫＢ／秒で書き込み、２．０ＭＢ／秒で読み出す。ＳＤＲＡＭは少なくとも１つの映像を記憶するのに使用することができ、また、ＤＯＳマシーンで行われるように大きなセクタ・カウントでコンパクトフラッシュへの書込みを試みることは高速書込みパフォーマンスを引き起こす。反対に、スマートメディアコントローラは５つのレジスタ設定を有する。すなわち、コマンドレジスタと、アドレス１レジスタと、アドレス２レジスタと、アドレス３レジスタと、データポートレジスタとである。これらの５つのレジスタはＡＲＭメモリ空間にマップされ、また、スマートメディアコントローラは異なるレジスタアクセス用の関連信号を自動的に生成する。

【０３２６】オーディオ入力／出力は、ＤＳＰバッファリングを備えたＩ／Ｏブロック１４０のシリアル・ポートを通る。赤外データ・アクセス（ＩｒＤＡ）は高速ＦＩＲコアとＩ／Ｏブロック１４０の一部とによって支援される。ブロック１４０はまた、ＡＧＣ利得および電子シャッターを調整するためのＣＣＤ／ＣＭＯＳイメージャ・モジュール制御，ＲＴＣ制御，適切なシステム応答のためにＡＲＭに内部中断を生成できるバッテリー電源検出，焦点およびズーム用のカメラレンズモーター制御，ユーザキーパッド入力，ＬＥＤインディケータ，フ

ラッシュライト制御および電源管理制御のようなアイテムを支援できる汎用入力／出力を含む。

【０３２７】ｉＭＸプログラミング

ＤＳＰ１２２は、ｉＭＸ１２４を指示して、ｉＭＸコマンドを送ることによってタスクを行わせる。これらのコマンドは、理解するのが複雑で、内部ループに固定される多くのパラメータを含む。ダイヤルモデルが別々のコマンド構成およびコマンド転送経路をＤＳＰプログラマーに提供し、その結果、コマンドがループ外に前もって構築され、一般データメモリがループ内で動くときにｉＭＸに転送される。通常用いられるｉＭＸコマンドは、Ｃ符号に前もってパッケージされており、プログラミングを容易としている。

【０３２８】ＡＲＭ／ＤＳＰタスク割り当て

ＡＲＭ１３０は、ウインドウズＣＥのようなオペレーティングシステムを走らせ、（コンパクトフラッシュカード（ＣＦＣ）へのような）低周波数，同期入力／出力を＊制御するとともに、やはり低速であるユーザ交流を制御し、また、すべての周辺モジュールは、プレビュー・エンジン，バーストモード圧縮，ＴＶ符号器，ＣＣＤコントローラ，ＵＳＢ，ＣＦ，ＩｒＤＡなどを制御する。ＤＳＰ１２２は、ＳＰＯＸのようなオペレーティングシステムを走らせ、すべてのリアルタイム機能（オートフォーカス，オート露光，オートホワイトバランス），リアルタイム入力／出力（オーディオＩＯ，モデムＩＯ），リアルタイムアプリケーション（例えば、オーディオプレイヤー），コンピュータを使用する高価な信号処理タスク（画像パイプライン，ＪＰＥＧ２０００，画像の縫合）を制御する。

【０３２９】集積回路チップのピンの説明

好ましい実施例のピンは以下の通りである。

【０３３０】

【表１１】

ＣＣＤセンサ

ピン総数：１６

１．Ｃ＿ＰＣＬＫ	（Ｉ）	画素クロック
２．Ｃ＿ＶＳＹＮＣ	（Ｉ／Ｏ）	垂直同期
３．Ｃ＿ＨＳＹＮＣ	（Ｉ／Ｏ）	水平同期
４．Ｃ＿ＦＩＥＬＤ	（Ｉ／Ｏ）	フィールドインディケータ
５．Ｃ＿ＷＥＮ	（Ｉ）	ＣＣＤＣ書込みイネーブル
６：１７．Ｃ＿ＤＡＴＡ	（Ｉ）	画像データ１２ビット

ＳＤＲＡＭインターフェース

ピン総数：５８

１．ＳＤＲ＿ＣＬＫ	（Ｏ）	マスタークロック
２．ＳＤＲ＿ＣＫＥ	（Ｏ）	クロックイネーブル
３．ＳＤＲ＿ＷＥ	（Ｏ）	書込みイネーブル
４．ＳＤＲ＿ＣＡＳ	（Ｏ）	列アドレスストローブ
５．ＳＤＲ＿ＲＡＳ	（Ｏ）	生アドレスストローブ
６．ＳＤＲ＿ＣＳ０	（Ｏ）	ＲＡＭの支援２ｐｃ
７．ＳＤＲ＿ＣＳ１	（Ｏ）	ＲＡＭの支援４ｐｃ
８：３９．ＤＱ[31：0]	（Ｉ／Ｏ）	データバス
４０：５４．ＳＤＲ＿Ａ[14：0]	（Ｏ）	アドレスバス
５５．ＳＤＲ＿ＤＱＭＨＨ	（Ｏ）	ＤＱ[31：24]用のＤＱＭＨ
５６．ＳＤＲ＿ＤＱＭＨＬ	（Ｏ）	ＤＱ[23：16]用のＤＱＭＨ
５７．ＳＤＲ＿ＤＱＭＬＨ	（Ｏ）	ＤＱ[15：8]用のＤＱＭＨ
５８．ＳＤＲ＿ＤＱＭＬＬ	（Ｏ）	ＤＱ[7：0]用のＤＱＭＨ

ＡＲＭバス

ピン総数：３９

１：２３．ＡＲＭ＿Ａ[22：0]	（Ｏ）	アドレスバス
２４：３９．ＡＲＭ＿Ｄ[15：0]	（Ｏ）	データバス

オーディオインターフェース

ピン総数：６

１．ＤＳＰ＿ＢＤＸ	（Ｏ）	シリアルポート送信
２．ＤＳＰ＿ＢＣＬＫＸ	（Ｉ／Ｏ）	送信クロック
３．ＤＳＰ＿ＢＦＳＸ	（Ｉ／Ｏ）	フレーム同期パルス
４．ＤＳＰ＿ＢＤＲ	（Ｉ）	シリアルデータ受信

77 78

５．ＤＳＰ＿ＢＣＬＫＲ　（Ｉ）　受信クロック
６．ＤＳＰ＿ＢＦＳＲ　（Ｉ）　フレーム同期パルス受信

外部フラッシュインターフェース
ピン総数：５
１．ＦＬＳＨ＿ＷＥ　（Ｏ）　書込みイネーブル
２．ＦＬＳＨ＿ＣＥ　（Ｏ）　チップ選択
３．ＦＬＳＨ＿ＯＥ　（Ｏ）　出力イネーブル
４．ＦＬＳＨ＿ＳＩＺＥ　（Ｉ）　８ビット／１６ビット選択
５．ＦＬＳＨ＿ＢＳＹ　（Ｉ）　ビジー入力

ＵＳＢ（Ｔ．Ｂ．Ｄ）
ピン総数：１０
１．Ｍ４８ＸＯ　（Ｏ）　４８ＭＨｚクロック出力
２．Ｍ４８ＸＩ　（Ｉ）　４８ＭＨｚクロック入力
３．ＵＳＢ＿ＤＰ　（Ｉ／Ｏ）　差分データ+
４．ＵＳＢ＿ＤＭ　（Ｉ／Ｏ）　差分データー
５．ＡＴＴＡＣＨ　（Ｉ）　付着検出

ＵＡＲＴ
ピン総数：５
１．ＲＸＤ　（Ｉ）　ＵＡＲＴ ＲＸ
２．ＴＸＤ　（Ｏ）　ＵＡＲＴ ＴＸ
３．ＥＲＸＤ　（Ｉ）　外部ＣＰＵ用のＵＡＲＴ Ｒｘ
４．ＥＴＸＤ　（Ｏ）　外部ＣＰＵ用のＵＡＲＴ ＴＸ
５．ＳＩＦＤＯ　（Ｏ）　シリアルＩ／Ｆデータ出力

ＩｒＤＡ
ピン総数：２
１．ＩＲＸＤ　（Ｉ）　ＩｒＤＡ ＲＸ
２．ＩＴＸＤ　（Ｏ）　ＩｒＤＡ ＴＸ

コンパクトフラッシュ
ピン総数：９
１．ＣＦＥ１　（Ｏ）　カードイネーブル＃１
２．ＣＦＥ２　（Ｏ）　カードイネーブル＃２
３．ＩＯＩＳ１６　（Ｏ）　Ｉ／Ｏ選択
４．ＳＴＳＣＨＧ　（Ｉ／Ｏ）　状態変化
５．ＣＦＷＡＩＴ　（Ｉ）　待機信号入力
６．ＣＦＲＳＴ　（Ｏ）　リセット
７．ＣＦＤ１　（Ｉ）　カード検出ピン＃１
８．ＣＦＤ２　（Ｉ）　カード検出ピン＃２
９．ＣＦＲＤＹ　（Ｉ）　レディ

ＴＶ／ＲＧＢ ＤＡＣアナログ出力
ピン総数：２７
１．ＩＲＥＦ（Ｒ）　（Ｉ）　Ｒ－ｃｈ電流基準制御
２．ＤＡＯＵＴ（Ｒ）　（Ｏ）　アナログ出力Ｒ－ｃｈ
３．ＧＮＤＡ　アナログＧＮＤ
４．ＶＣＣＡ　アナログＶＣＣ
５．ＢＩＡＳ　（Ｉ）　位相補償ｃａｐ．Ｒ－ｃｈ
６．ＶＲＥＦ　（Ｉ）　ＲＧＢ共通基準電圧
７．ＩＲＥＦ（Ｇ）　（Ｉ）　Ｇ－ｃｈ電流基準制御
８．ＤＡＯＵＴ（Ｇ）　（Ｏ）　アナログ出力Ｇ－ｃｈ
９．ＧＮＤＡ　アナログＧＮＤ
１０．ＶＣＣＡ　アナログＶＣＣ

79 80

１１．ＢＩＡＳ	（Ｉ）	位相補償ｃａｐ．Ｇ－ｃｈ
１２．ＩＲＥＦ（Ｂ）	（Ｉ）	Ｂ－ｃｈ電流基準制御
１３．ＤＡＯＵＴ（Ｂ）	（Ｏ）	アナログ出力Ｂ－ｃｈ
１４．ＧＮＤＡ		アナログＧＮＤ
１５．ＶＣＣＡ		アナログＶＣＣ
１６．ＢＩＡＳ	（Ｉ）	位相補償ｃａｐ．Ｂ－ｃｈ
１７．ＩＲＥＦ（Ｃ）	（Ｉ）	コンポジット電流基準制御
１８．ＤＡＯＵＴ（Ｃ）	（Ｏ）	アナログ出力コンポジット
１９．ＧＮＤＡ		アナログＧＮＤ
２０．ＶＣＣＡ		アナログＶＣＣ
２１．ＶＲＥＦ	（Ｉ）	コンポジット基準電圧
２２．ＢＩＡＳ	（Ｉ）	位相補償ｃａｐ．コンポジット
２３．ＤＶＣＣ		ＤＡＣ用のデジタルＶＣＣ
２４．ＤＧＮＤ		ＤＡＣ用のデジタルＧＮＤ
２５．ＨＳＹＮＣ	（Ｏ）	ＲＧＢ出力用のＨ同期出力
２６．ＶＣＳＹＮＣ	（Ｏ）	Ｖ同期／コンポジット同期（レジスタによって選択）

ＧＩＯ

ピン総数：３２［３１：０］

１：３２：ＧＩＯ	（Ｉ／Ｏ）	汎用Ｉ／Ｏ

その他

ピン総数：１５

１．ＲＥＳＥＴ	（Ｉ）	リセットをパワー・オン
２．Ｍ２７ＸＩ	（Ｉ）	２７ＭＨｚ入力
３．Ｍ２７ＸＯ	（Ｏ）	２７ＭＨｚ出力
４．ＴＣＫ	（Ｉ）	ＪＴＡＧクロック
５．ＴＤＩ	（Ｉ）	ＪＴＡＧデータ入力
６．ＴＤＯ	（Ｏ）	ＪＴＡＧデータ出力
７．ＴＭＳ	（Ｉ）	ＪＴＡＧテストモード選択
８．ＴＲＳＴ	（Ｉ）	ＪＴＡＧテストリセット
９．ＥＭＵ０	（Ｉ／Ｏ）	エミュレータ割込み０ピン
１０．ＥＭＵ１	（Ｉ／Ｏ）	エミュレータ割込み１ピン
１１．ＴＥＳＴ０	（Ｉ）	テスト入力０
１２．ＴＥＳＴ１	（Ｉ）	テスト入力１
１３．ＳＣＡＮ	（Ｉ）	テスト入力
１４．ＴＥＳＴＳＬ０	（Ｉ）	テスト・モード選択０
１５．ＴＥＳＴＳＬ１	（Ｉ）	テスト・モード選択１

合計ピン総数

ＣＣＤセンサ	１７
ＳＤＲＡＭ Ｉ／Ｆ	５８
ＡＲＭバス	３９
オーディオＩ／Ｆ	６
フラッシュメモリＩ／Ｆ	５
ＵＳＢ	５
ＵＡＲＴ	５
ＩｒＤＡ	２
コンパクトフラッシュＩ／Ｆ	９
４ＤＡＣ	２６
ＧＩＯ	３２
その他	１５

サブ合計　２１９ピン
電源　３７ピン（１４％）
合計　２５６ピン

【０３３１】オーディオプレイヤー

携帯デジタルオーディオプレイヤーは、最も人気のある消費者用の製品の１つであることが期待される。現在、ＭＰＥＧ－１レイヤー３オーディオ圧縮標準に基づいたＭＰ－３プレイヤーは携帯オーディオ市場で急速に成長しているが、ＭＰＥＧ－２ＡＡＣおよびＤｏｂｙＡＣ－３が、標準として定着しつつあると考えられる代替デジタルオーディオ符号化フォーマットである。このように、好ましい実施例のプログラム可能性は、デジタルオーディオプレイヤーの機能を含ませることを可能とする。オーディオは、フラッシュメモリやＰＣなどを介して入力することができ、また、復号されたものはシリアルポートで出力することができる。復号化プログラムは、フラッシュメモリやＲＯＭなどからロードすることができる。

【図面の簡単な説明】

【図１ａ】機能ブロック・フォーマットの好ましい実施例システムを示す。

【図１ｂ】機能ブロック・フォーマットの好ましい実施例システムを示す。

【図２】データ・フローを例示する。

【図３ａ】データ・フローを例示する。

【図３ｂ】データ・フローを例示する。

【図４】データ・フローを例示する。

【図５】データ・フローを例示する。

【図６】データ・フローを例示する。

【図７ａ】ＣＦＡ配列を示す。

【図７ｂ】ＣＦＡ配列を示す。

【図８】ホワイトバランスのための機能図である。

【図９ａ】ガンマ補正を示す。

【図９ｂ】ガンマ補正を示す。

【図９ｃ】ガンマ補正を示す。

【図１０ａ】ＣＦＡ補間を例示する。

【図１０ｂ】ＣＦＡ補間を例示する。

【図１０ｃ】ＣＦＡ補間を例示する。

【図１０ｄ】ＣＦＡ補間を例示する。

【図１０ｅ】ＣＦＡ補間を例示する。

【図１０ｆ】ＣＦＡ補間を例示する。

【図１０ｇ】ＣＦＡ補間を例示する。

【図１０ｈ】ＣＦＡ補間を例示する。

【図１０ｉ】ＣＦＡ補間を例示する。

【図１０ｊ】ＣＦＡ補間を例示する。

【図１０ｋ】ＣＦＡ補間を例示する。

【図１０ｌ】ＣＦＡ補間を例示する。

【図１１ａ】カラー変換を示す。

【図１１ｂ】カラー変換を示す。

【図１２ａ】メモリ・コントローラ・データ・フローを示す。

【図１２ｂ】メモリ・コントローラ・データ・フローを示す。

【図１３ａ】バースト圧縮／圧縮解除エンジンの機能ブロック図である。

【図１３ｂ】バースト圧縮／圧縮解除エンジンの機能ブロック図である。

【図１４】プレビュー・エンジンの機能ブロック図である。

【図１５】オンスクリーン・ディスプレイ・ブロック図である。

【図１６】オンスクリーン・ディスプレイ・ウインドウである。

【図１７】ハードウェア・カーソルを示す。

【図１８ａ】ＤＳＰサブシステムを示す。

【図１８ｂ】ＤＳＰサブシステムを示す。

【図１９】並列乗算－累算データパスを示す。

【図２０】コプロセッサ・アーキテクチャを示す。

【図２１】ルックアップ・テーブル・加速器を例示する。

【図２２】可変長符号器のブロック図である。

【図２３ａ】ブリッジを示す。

【図２３ｂ】ブリッジを示す。

【図２３ｃ】ブリッジを示す。

【図２４】マルチプロセッサ・デバグ支持を示す。

【図２５】ＵＡＲＴ接続を例示する。

【図２６】フラッシュカード／スマートカード・インターフェースのブロック図である。

【図２７】カラーフィルタ・アレイ補間を例示する。

【図２８】カラーフィルタ・アレイ補間を例示する。

【図２９】カラーフィルタ・アレイ補間を例示する。

【図３０】カラーフィルタ・アレイ補間を例示する。

【図３１】カラーフィルタ・アレイ補間を例示する。

【図３２】カラーフィルタ・アレイ補間を例示する。

【図３３ａ】カラーフィルタ・アレイ補間を例示する。

【図３３ｂ】カラーフィルタ・アレイ補間を例示する。

【図３４】カラーフィルタ・アレイ補間を例示する。

【図３５ａ】カラーフィルタ・アレイ補間を例示する。

【図３５ｂ】カラーフィルタ・アレイ補間を例示する。

【図３６ａ】カラーフィルタ・アレイ補間を例示する。

【図３６ｂ】カラーフィルタ・アレイ補間を例示する。

【図３７ａ】カラーフィルタ・アレイ補間を例示する。

【図３７ｂ】カラーフィルタ・アレイ補間を例示する。

【図３８】カラーフィルタ・アレイ補間を例示する。

【図３９ａ】ホワイトバランシングを示す。

【図３９ｂ】ホワイトバランシングを示す。

【図４０】ホワイトバランシングを示す。

83

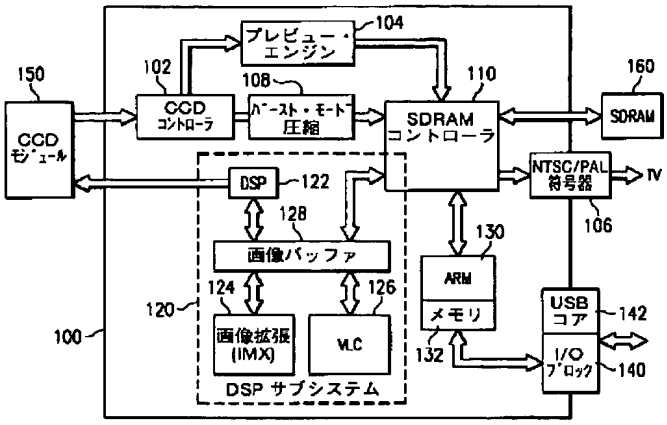
【図４１ａ】画像サイズ調整を指示する。
【図４１ｂ】画像サイズ調整を指示する。
【図４２ａ】画像のサイズ調整を指示する。
【図４２ｂ】画像サイズ調整を指示する。
【図４２ｃ】画像サイズ調整を指示する。
【図４２ｄ】画像サイズ調整を指示する。
【図４２ｅ】画像サイズ調整を指示する。
【図４３】色調スケーリングを例示する。
【図４４】色調スケーリングを例示する。
【図４５】色調スケーリングを例示する。

84

＊【図４６ａ】同期を示す。
【図４６ｂ】同期を示す。
【図４７】同期を示す。
【図４８】同期を示す。
【図４９】復号化バッファリングを示す。
【図５０】復号化バッファリングを示す。
【図５１ａ】復号化バッファリングを示す。
【図５１ｂ】復号化バッファリングを示す。
【図５２】復号化バッファリングを示す。

＊10

【図１ａ】



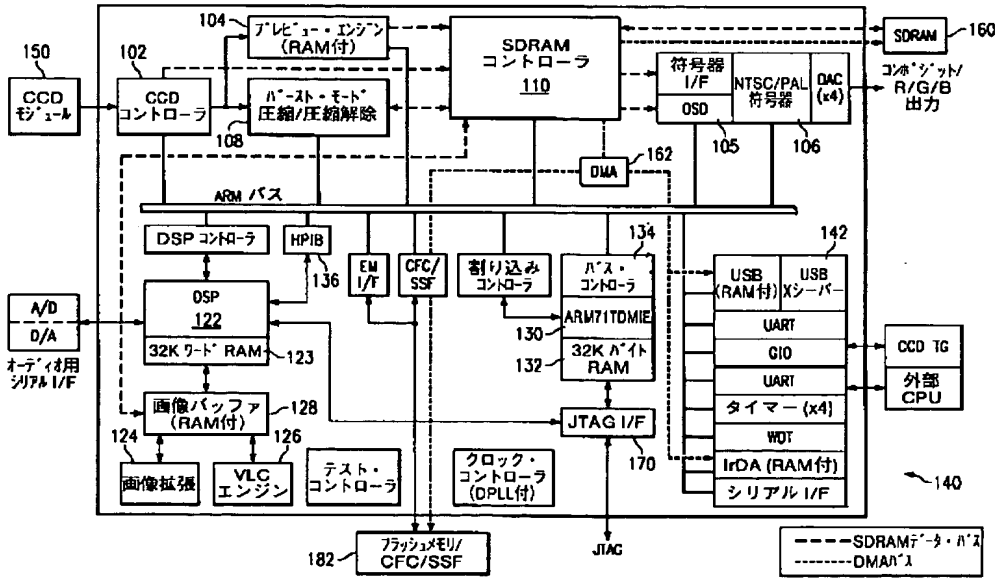
【図７ａ】

R	G	R	G
G	B	G	B
R	G	R	G
G	B	G	B

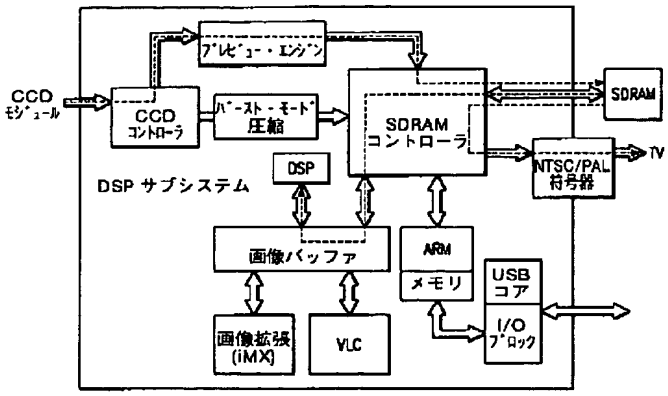
【図７ｂ】

Ye	Cy	Ye	Cy
G	Mg	G	Mg
Ye	Cy	Ye	Cy
G	Mg	G	Mg

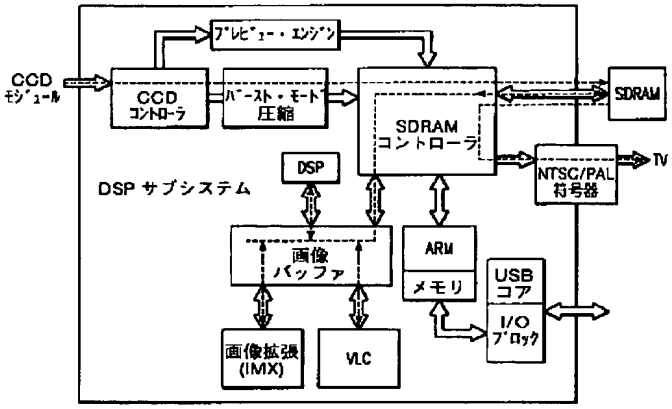
【図１ｂ】



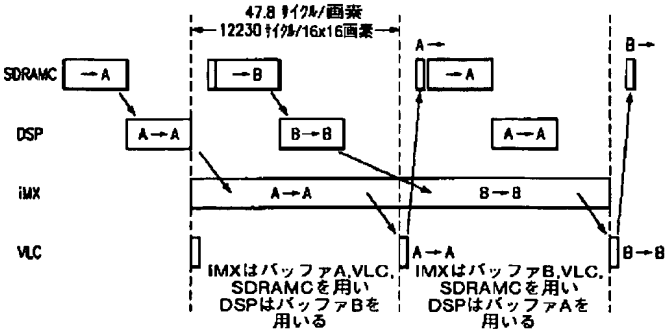
【図２】



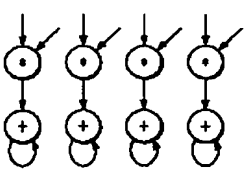
【図３ａ】



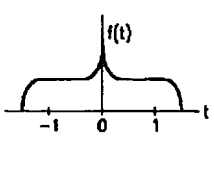
【図３ｂ】



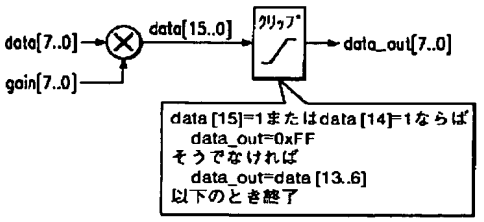
【図１９】



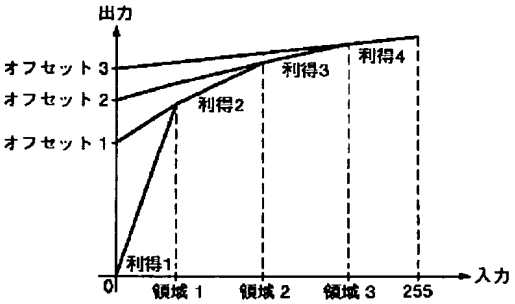
【図４１ｂ】



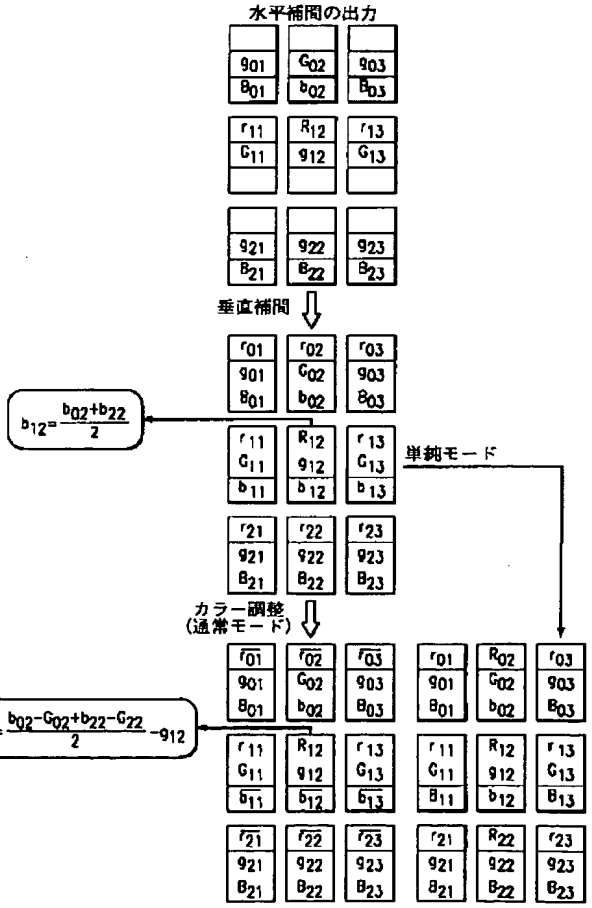
【図９ａ】



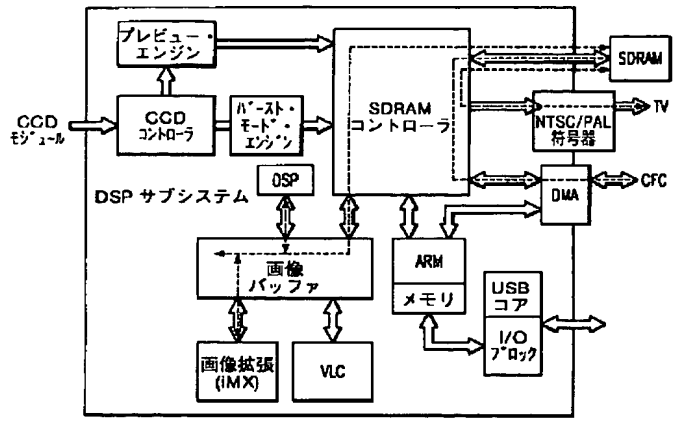
【図９ｂ】



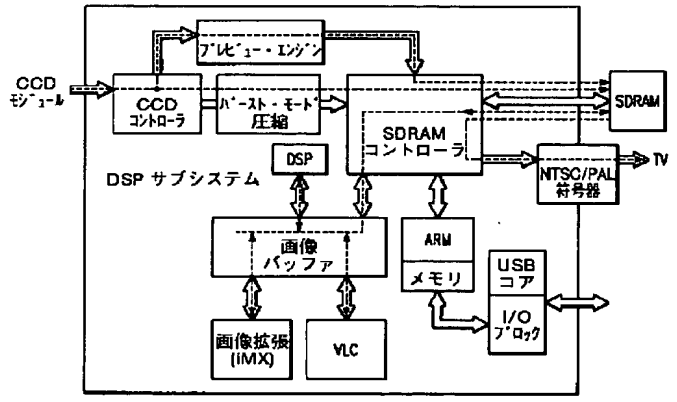
【図１０ｅ】



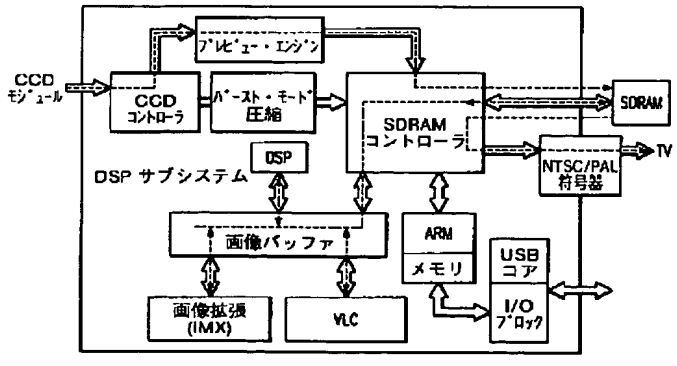
【図４】



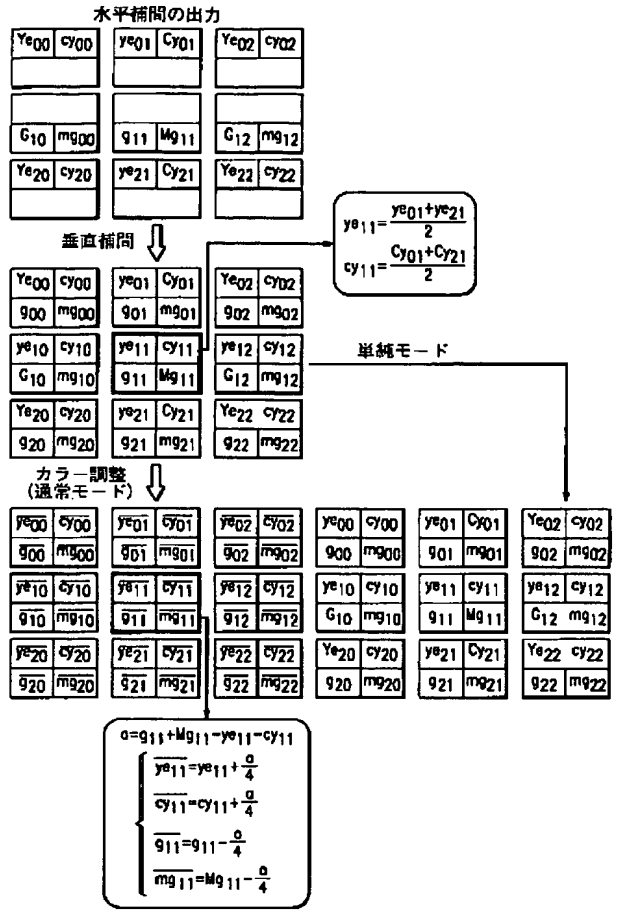
【図５】



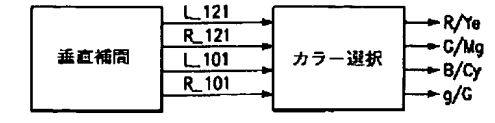
【図６】



【図１０ｆ】



【図１０ｉ】



ノイズフィルタ=オフ

$R/Ye=R_{12}$

$G/Mg=g_{12}$

$g/Cy=\frac{G_{02}+G_{22}}{2}$

$B/Cy=\frac{b_{02}+b_{22}}{2}$

ノイズフィルタ=オン

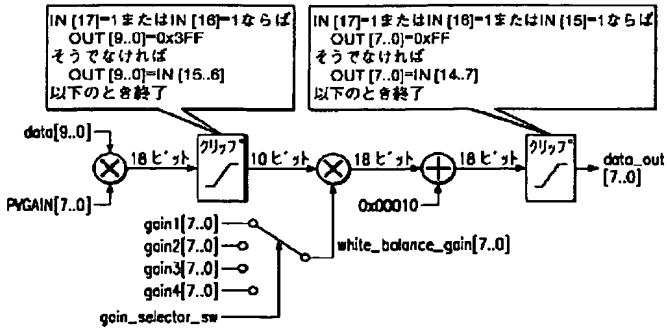
$R/Ye=R_{12}-g_{12}+\frac{G_{02}+2g_{12}+G_{22}}{4}$

$G/Mg=\frac{G_{02}+2g_{12}+G_{22}}{4}$

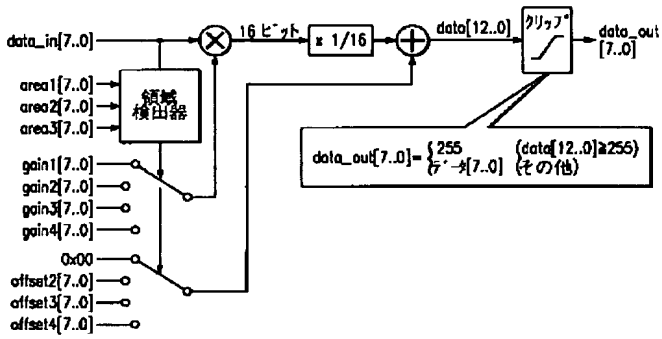
$g/Cy=\frac{G_{02}+G_{22}}{2}$

$B/Cy=\frac{b_{02}+b_{22}}{2}$

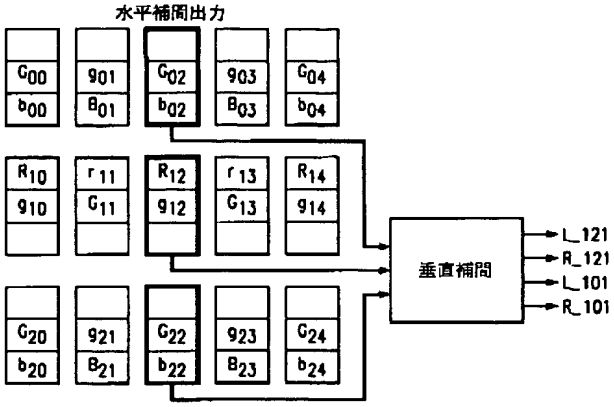
【図８】



【図９ｃ】



【図１０ｇ】



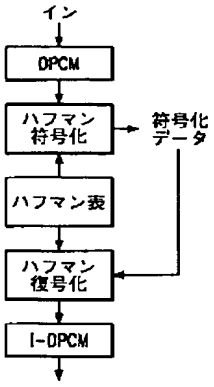
ノイズフィルタ=オフ

$$\begin{cases} L\_121 = R_{12} \\ R\_121 = g_{12} \\ L\_101 = \dfrac{G_{02}+G_{22}}{2} \\ R\_101 = \dfrac{b_{02}+b_{22}}{2} \end{cases}$$

ノイズフィルタ=オン

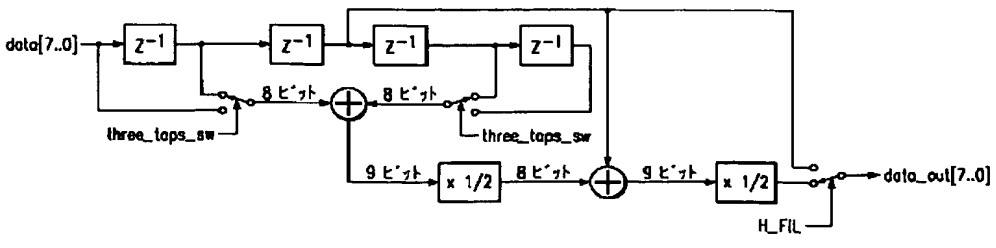
$$\begin{cases} L\_121 = R_{12} - g_{12} + \dfrac{G_{02}+2g_{12}+G_{22}}{4} \\ R\_121 = \dfrac{G_{02}+2g_{12}+G_{22}}{4} \\ L\_101 = \dfrac{G_{02}+G_{22}}{2} \\ R\_101 = \dfrac{b_{02}+b_{22}}{2} \end{cases}$$

【図１３ｂ】

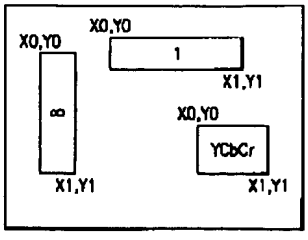


【図１０ａ】

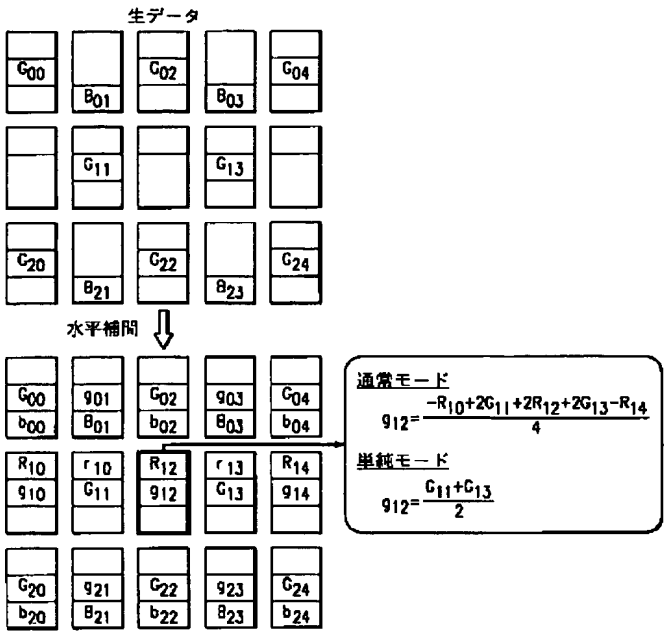
【図１０ｂ】



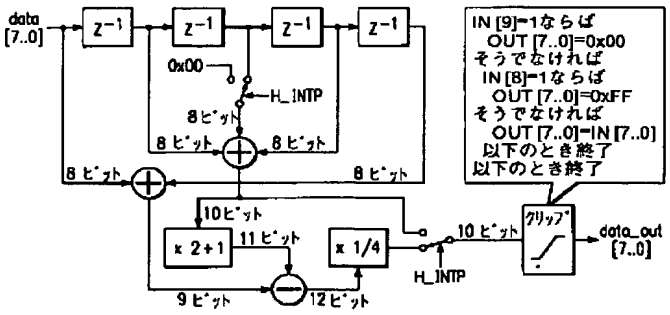
【図１６】



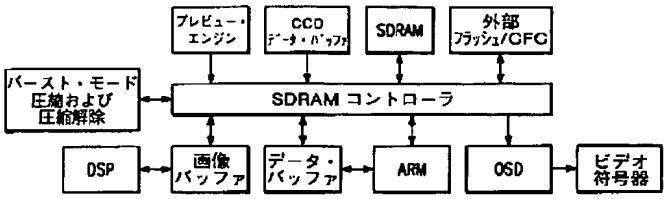
【図１０ｃ】



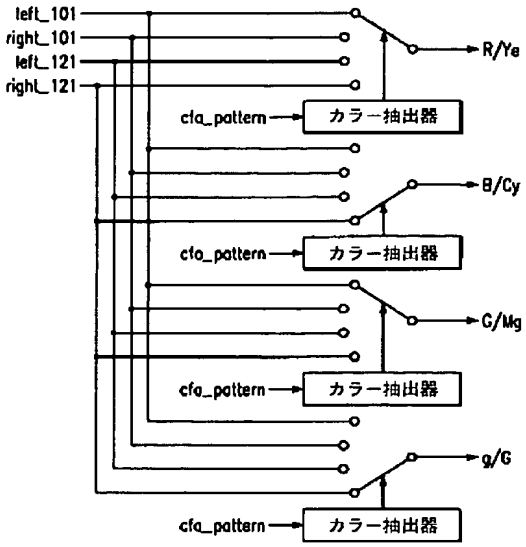
【図１０ｄ】



【図１２ａ】

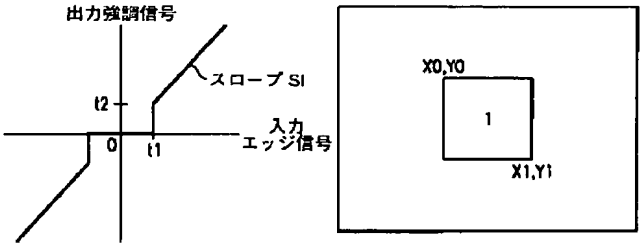


【図１０ｊ】

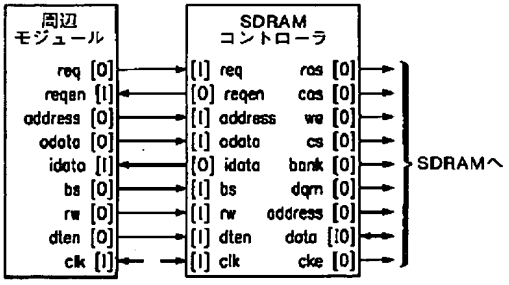


【図１１ｂ】

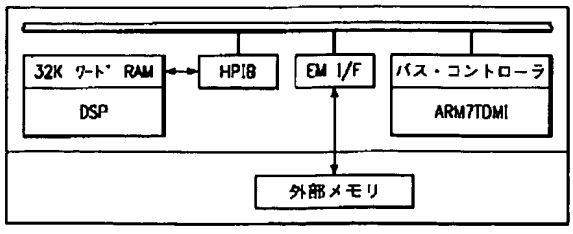
【図１７】



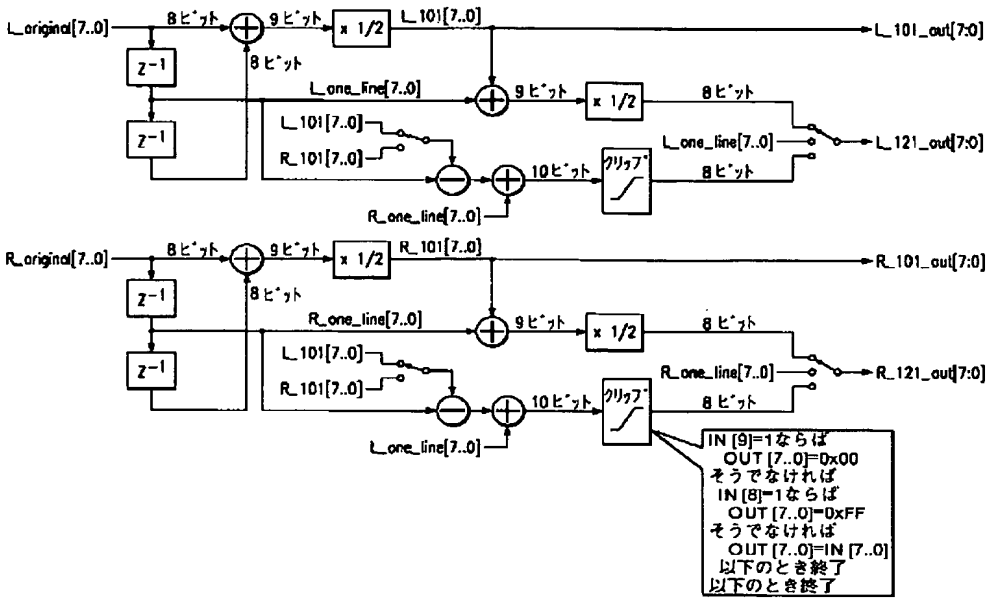
【図１２ｂ】



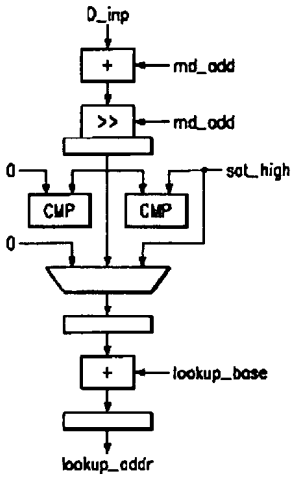
【図２３ａ】



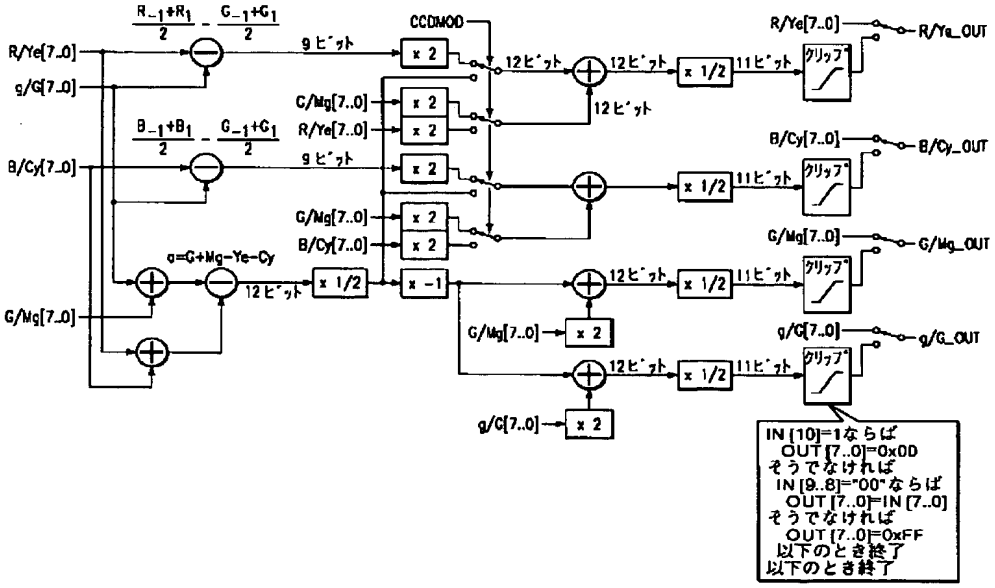
【図１０ｈ】



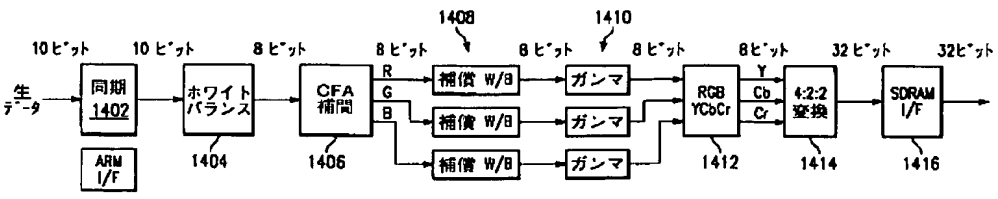
【図２１】



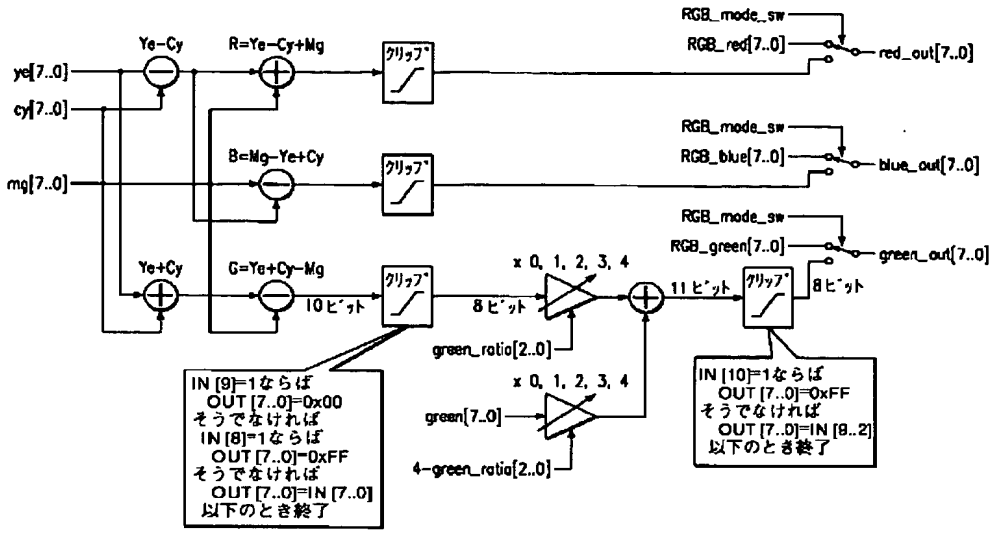
【図１０ｋ】



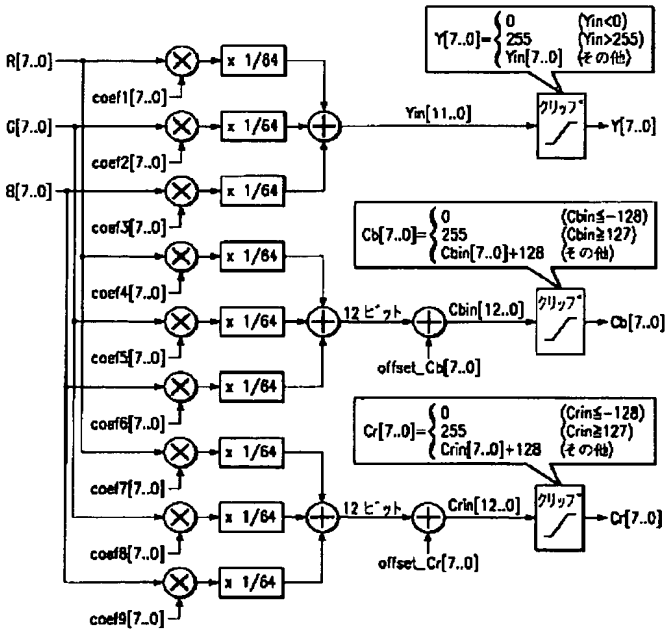
【図１４】



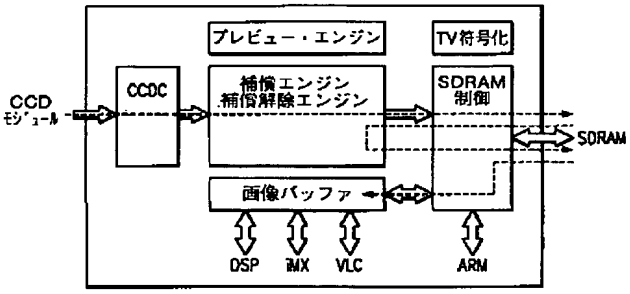
【図１０１】



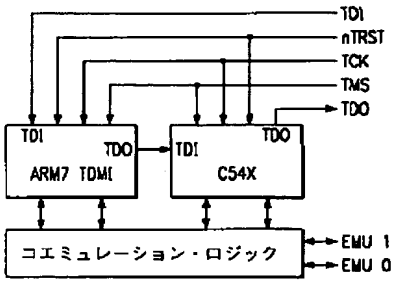
【図１１ａ】



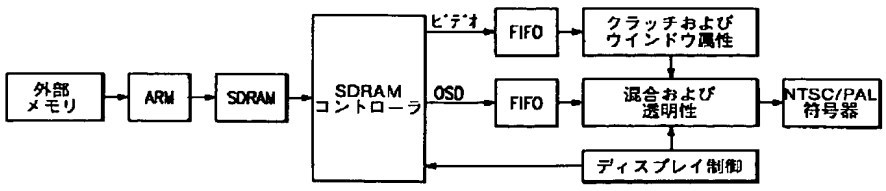
【図１３ａ】



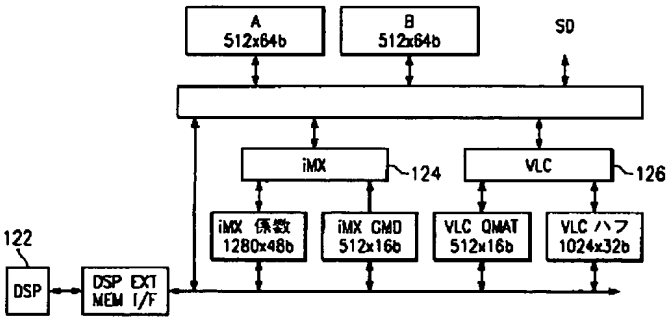
【図２４】



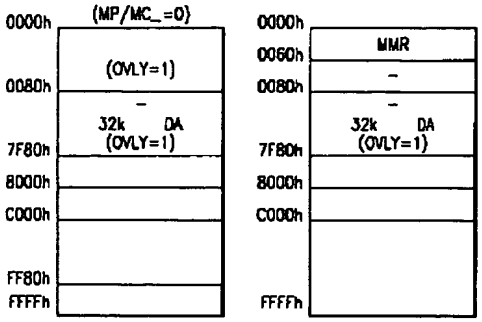
【図１５】



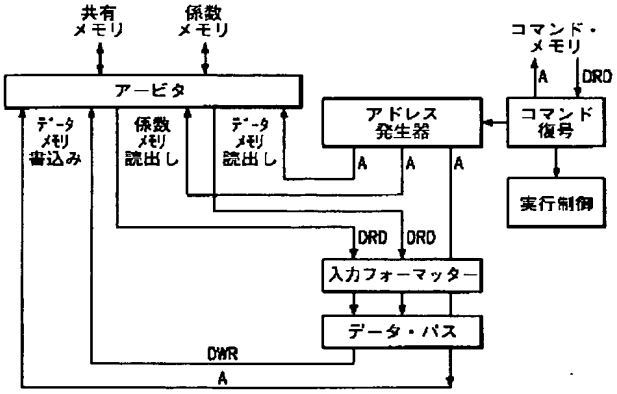
【図１８ａ】



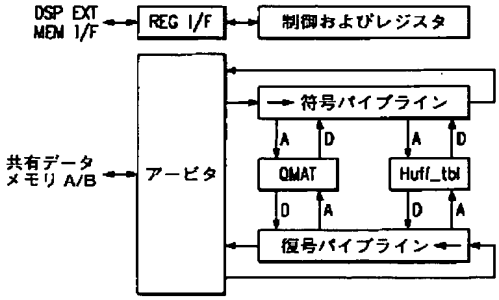
【図１８ｂ】



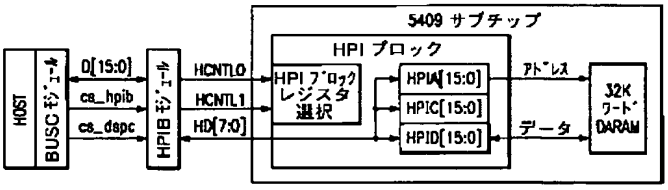
【図２０】



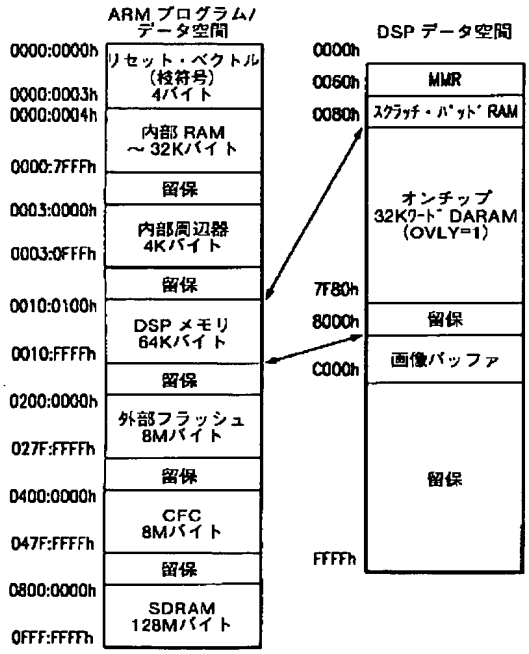
【図２２】



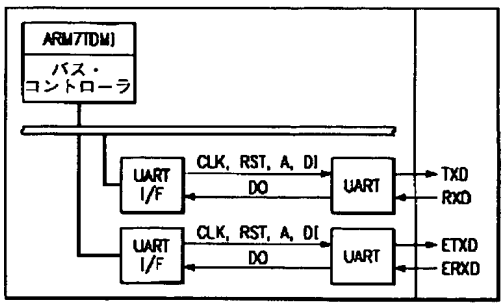
【図２３ｂ】



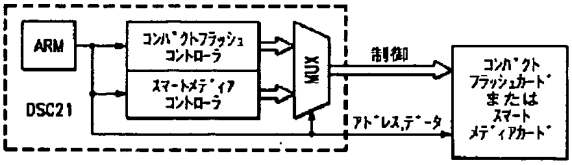
【図２３ｃ】



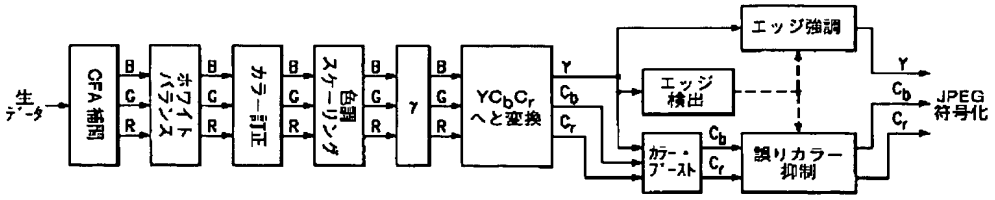
【図２５】



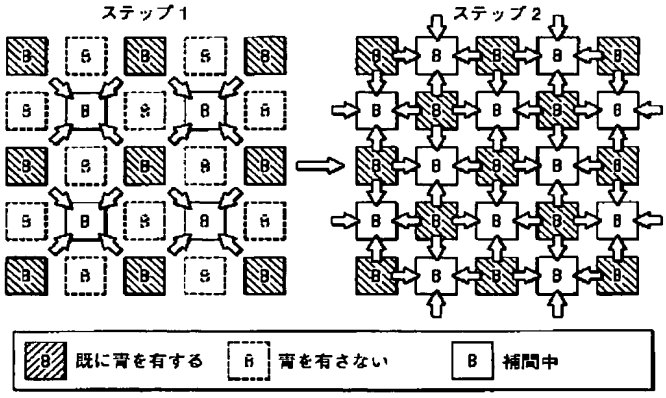
【図２６】



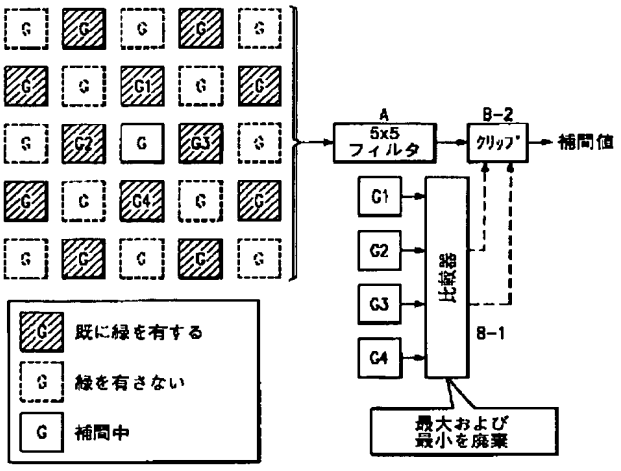
【図２７】



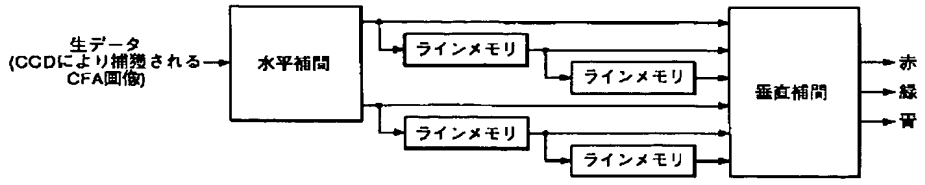
【図２８】



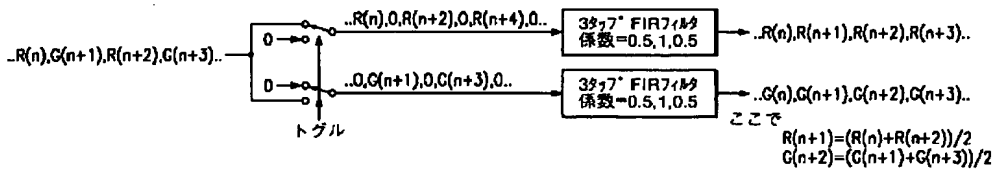
【図２９】



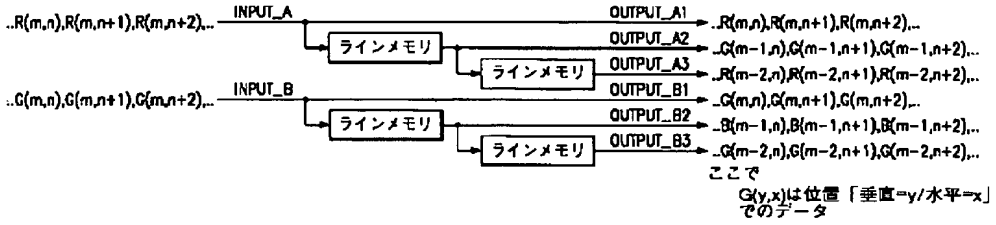
【図３０】



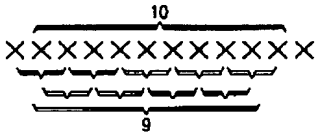
【図３１】



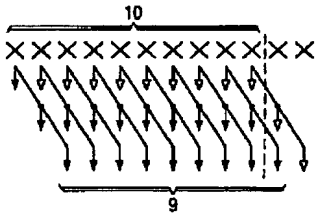
【図３２】



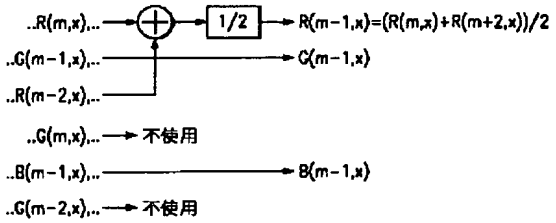
【図４２ａ】



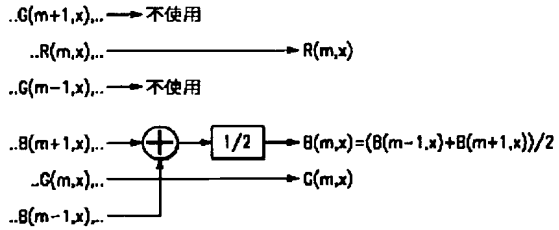
【図４２ｂ】



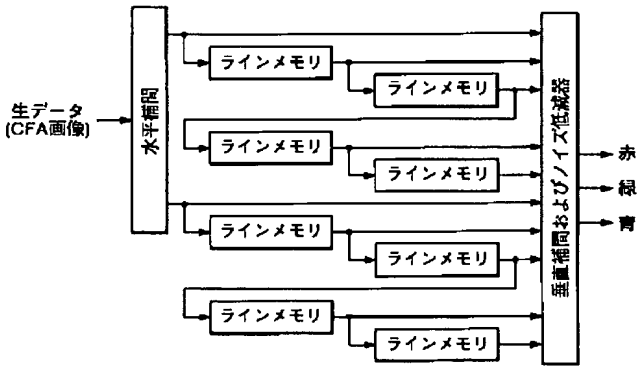
【図３３ａ】



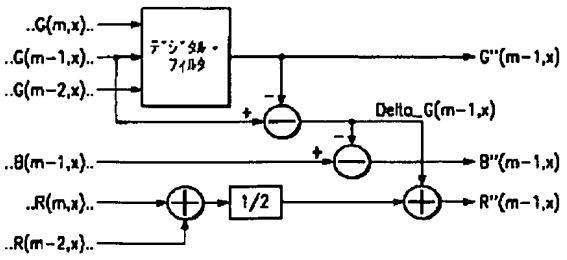
【図３３ｂ】



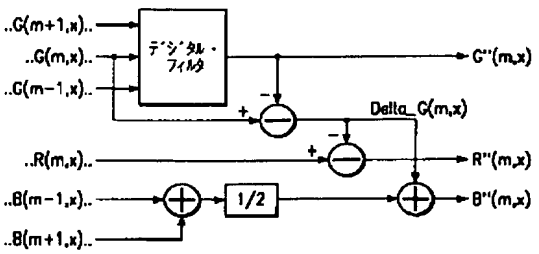
【図３４】



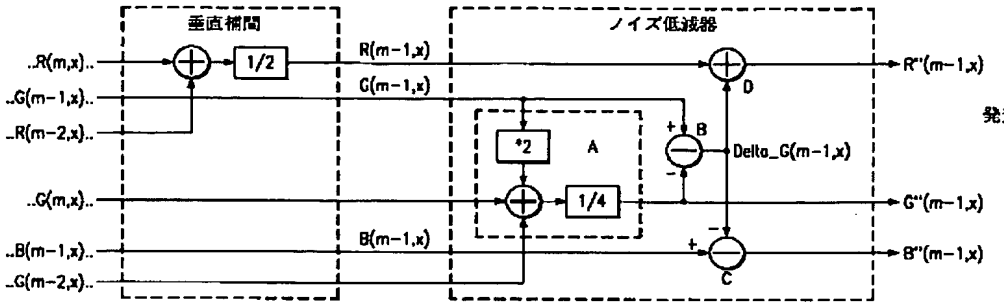
【図３７ａ】



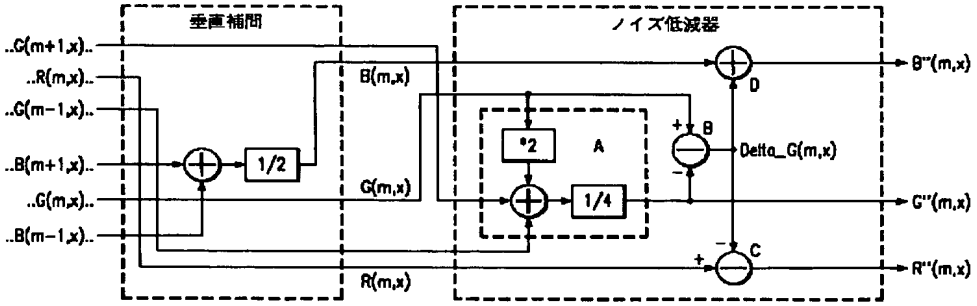
【図３７ｂ】



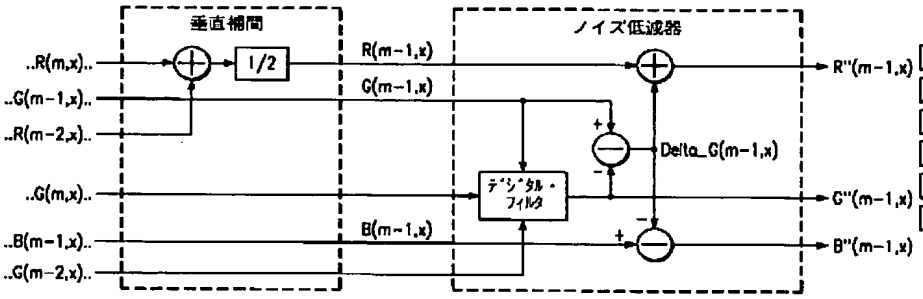
【図３５ａ】



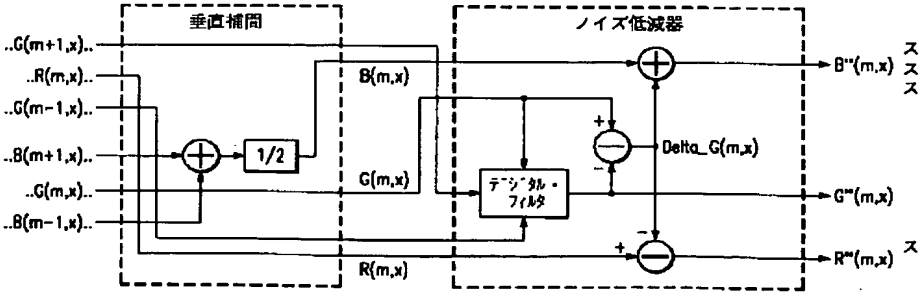
【図３５ｂ】



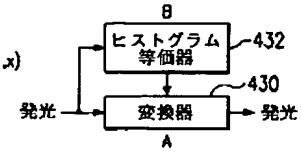
【図３６ａ】



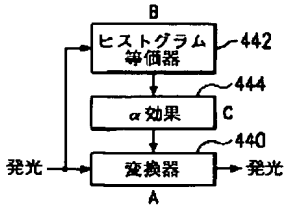
【図３６ｂ】



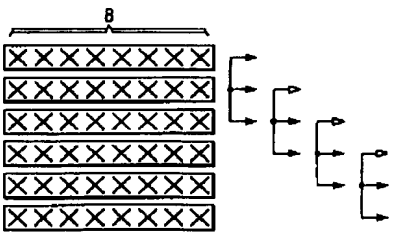
【図４３】



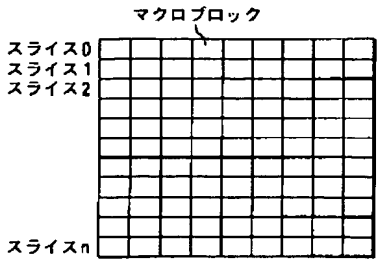
【図４４】



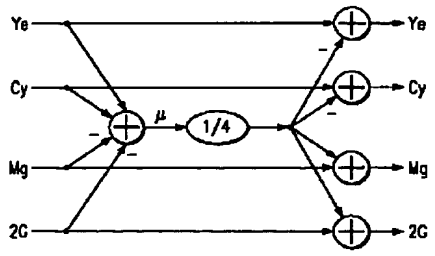
【図４２ｅ】



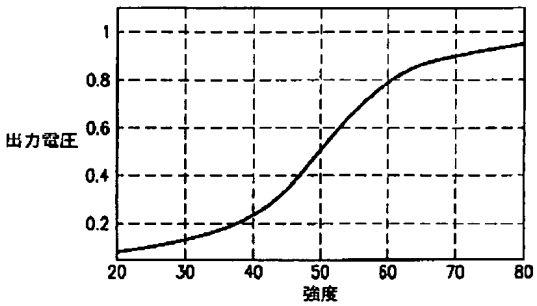
【図４９】



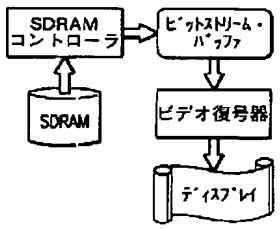
【図３８】



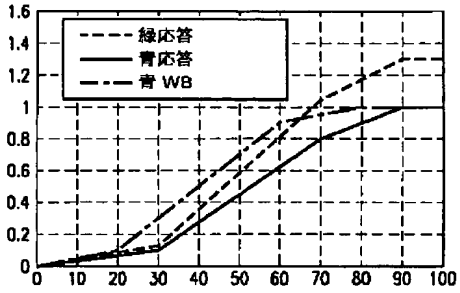
【図３９ａ】



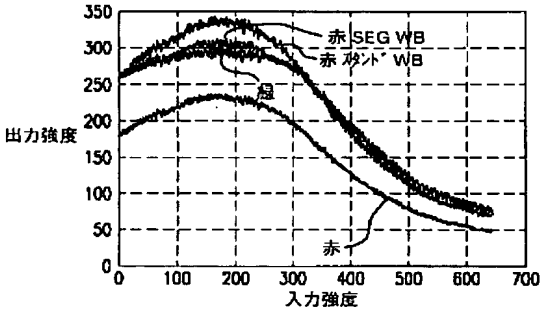
【図５０】



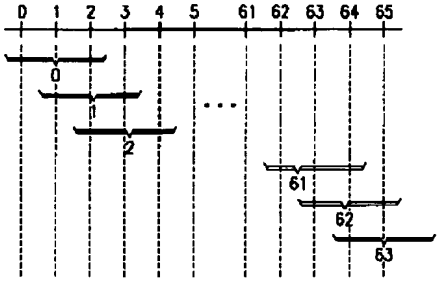
【図３９ｂ】



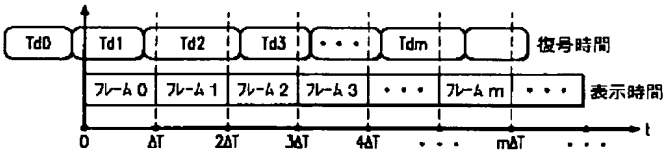
【図４０】



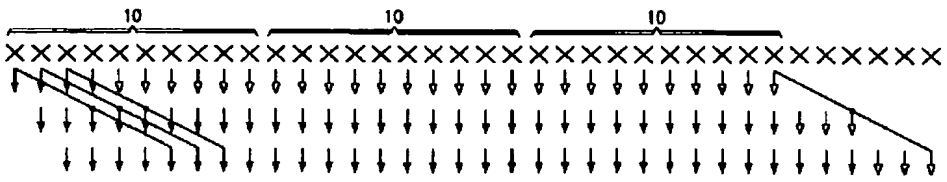
【図４１ａ】



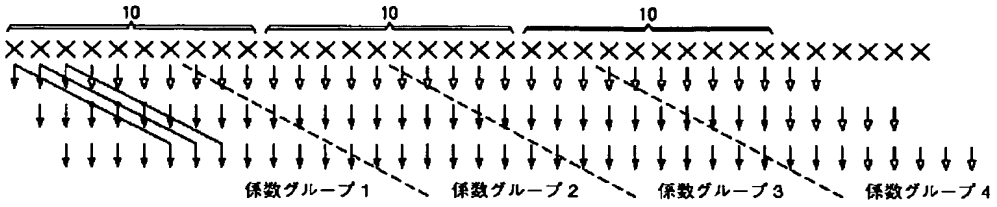
【図４６ａ】



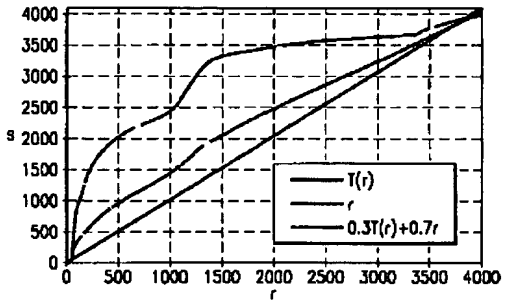
【図４２ｃ】



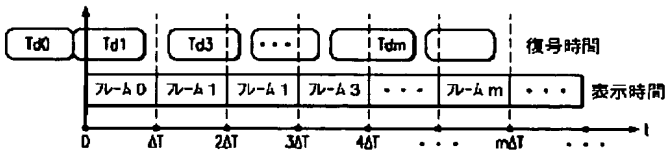
【図４２ｄ】



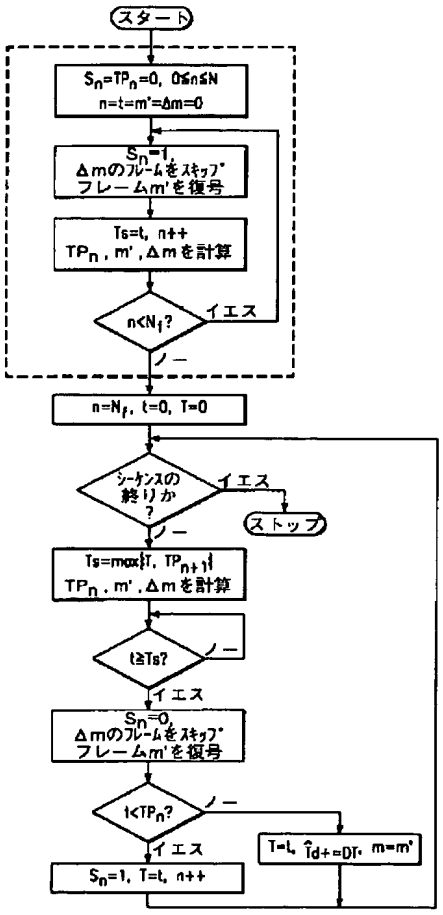
【図４５】



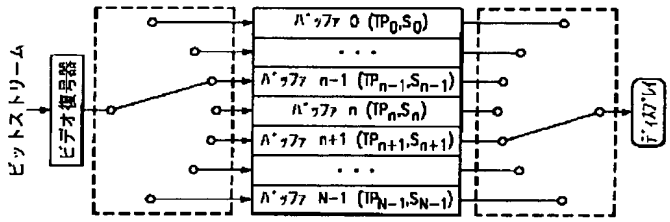
【図４６ｂ】



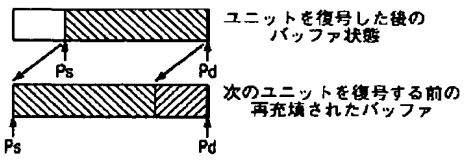
【図４８】



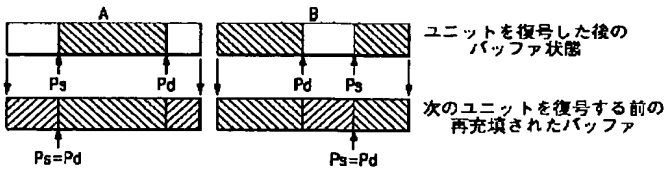
【図４７】



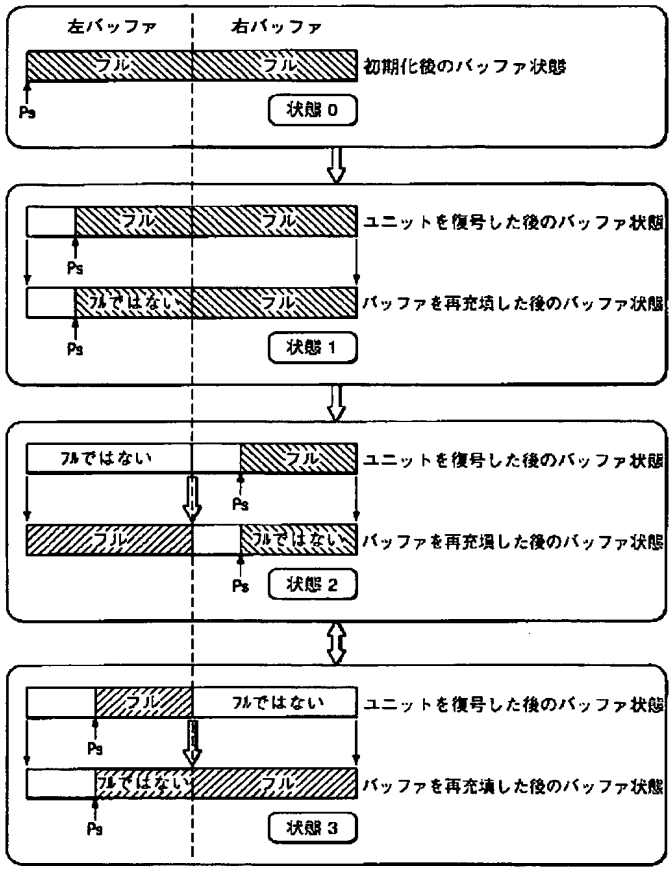
【図５１ａ】



【図５１ｂ】



【図５２】



フロントページの続き

(51) Int. Cl.[7]	識別記号	FI		テーマコード'(参考)
Ｈ０４Ｎ	1/393	Ｈ０４Ｎ	1/393	５Ｃ０６５
	1/48		5/232	Ｚ　５Ｃ０７６
	5/232		9/04	Ｂ　５Ｃ０７９
	5/91		101:00	
	7/24		9/79	Ｇ
	9/04		5/91	Ｊ
	9/79		1/46	Ａ
// Ｈ０４Ｎ	101:00		7/13	Ｚ

(31)優先権主張番号　６３２５４３
(32)優先日　平成12年８月４日(2000．８．４)
(33)優先権主張国　米国（ＵＳ）
(72)発明者　アキラ　オサモト
茨城県稲敷郡東町幸田972－２
(72)発明者　小柴　修
茨城県つくば市高野台３－11－２－402
(72)発明者　山内　暁
茨城県土浦市乙戸1022－17
(72)発明者　ミンファ　ゾウ
アメリカ合衆国　テキサス、プラノ、　オハイオ　ドライブ　2523、ナンバー108
(72)発明者　クラウス　イルグナー
ドイツ連邦共和国　ミュンヘン、　マルクアルトシュタイナー　シュトラーセ　12

(72)発明者　ラジェンドラ　タルリ
アメリカ合衆国　テキサス、プラノ、　ファンテン　ヘッド　ドライブ　2220
(72)発明者　ヤウンジュン　ヨー
アメリカ合衆国　テキサス、リチャードソン、　ダブリュ、レンナー　ロード　800、ナンバー1418
(72)発明者　ジー　リアン
アメリカ合衆国　テキサス、プラノ、　フロステッド　グリーン　レーン　2505
(72)発明者　マンディ゛ツァイ
台湾、台北、ファ　リン　ストリート　72、４エフ
(72)発明者　キヨシ　ツノダ
神奈川県横浜市港北区篠原台町36－33
(72)発明者　シンリ　イナモリ
神奈川県横浜市金沢区釜利谷東２－18－22－404

Ｆターム(参考)　5B057 BA02 CA01 CA08 CA12 CA16 CB01 CB08 CB12 CB16 CC01 CD06 CE17 CE18 CH02 CH08 CH09 CH14
5C022 AA13 AB00 AC69
5C053 FA08 GB36 KA01 KA21 KA24 KA25 KA26
5C055 AA07 BA06 EA01 GA00 HA18 HA31 HA35 HA37
5C059 KK50 LB11 ME05 ME17 PP15 PP16 SS11 UA02 UA11 UA34 UA38
5C065 AA03 BB02 GG13 GG15 GG18 GG29 GG31 GG32
5C076 AA21 AA26 BA05 BA06 BB04
5C079 HB01 HB04 JA23 LA17 LA23 LA28 LA37 LB02 MA02 MA11 NA03 NA09

【外国語明細書】

DIGITAL STILL CAMERA SYSTEM AND METHOD

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority from provisional applications Serial Nos. 60/172,780, filed 12/20/99; 60/176,272, filed 1/14/00; 60/177,432, filed 1/21/00; 60/214,951, filed 06/29/00; and 60/215,000, filed 06/29/00. The following pending US patent applications disclose related subject matter and have a common assignee with the present application: Serial No. , filed ... .

BACKGROUND OF THE INVENTION

This invention relates to integrated circuits, and more particularly, to integrated circuits and methods for use with digital cameras.

Recently, Digital Still Cameras (DSCs) have become a very popular consumer appliance appealing to a wide variety of users ranging from photo hobbyists, web developers, real estate agents, insurance adjusters, photo-journalists to everyday photography enthusiasts. Recent advances in large resolution CCD arrays coupled with the availability of low-power digital signal processors (DSPs) has led to the development of DSCs that come quite close to the resolution and quality offered by traditional film cameras. These DSCs offer several additional advantages compared to traditional film cameras in terms of data storage, manipulation, and transmission. The digital representation of captured images enables the user to easily incorporate the images into any type of electronic media and transmit them over any type of network. The ability to instantly view and selectively store captured images provides the flexibility to minimize film waste and instantly determine if the image needs to be captured again. With its digital representation the image can be corrected, altered, or modified after its capture. See for example, Venkataraman et al, "Next Generation Digital Camera Integration and Software Development Issues" in Digital Solid State Cameras: Design and Applications, 3302 Proc. SPIE (1998). Similarly, USP 5,528,293 and USP 5,412,425 disclose aspects of digital still

camera systems including storage of images on memory cards and power conservation for battery-powered cameras.

SUMMARY OF THE INVENTION

The invention provides a digital still camera architecture with features selected from programmable camera functions plus short audio-video clip capabilities, dual processors plus an image coprocessor, burst mode compression/decompression engine, programmable preview engine, and integration of all camera peripherals including IrDA, USB, NTSC/PAL encoder, DACs for RGB, UART, and compact flash card/smart media card interface.

This has advantages including flexibility, adaptability, and efficiency.

BRIEF DESCRIPTION OF THE DRAWINGS

Figures 1a-1b show a preferred embodiment system in functional block format.

Figures 2-6 illustrate data flows.

Figures 7a-7b show CFA arrangements.

Figure 8 is a functional diagram for white balance.

Figures 9a-9b show gamma correction.

Figure 10 illustrates CFA interpolation.

Figures 11a-11b show color conversion.

Figures 12a-12b show a memory controller data flow.

Figure 13 is a functional block diagram of a burst compression/-decompression engine.

Figure 14 is a functional block diagram of a preview engine.

Figure 15 is an on screen display block diagram.

Figure 16 is an on screen display window.

Figure 17 shows a hardware cursor.

Figure 18 illustrates a DSP subsystem.

Figure 19 shows parallel multiply-accumulate datapath.

Figure 20 shows a coprocessor architecture.

Figure 21 illustrates a look-up table accelerator.

Figure 22 is a block diagram of a variable length coder.

Figure 23 shows a bridge.

Figure 24 shows multiprocessor debugging support.

Figure 25 illustrates UART connections.

Figure 26 is a block diagram of flash card/smart card interface.

Figures 27-38 illustrate color filter array interpolations.

Figures 39a-39b and 40 show white balancing.

Figures 41a-41b and 42a- 42e indicate image resizing.

Figures 43-45 illustrate tone-scaling.

Figures 46a-46b and 47-48 show synchronization.

Figures 49-52 show decoding buffering.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

System overview

Figures 1a-1b show the various high-level functional blocks in a preferred embodiment digital still camera (DSC) and systems with Figure 1b providing more detail than Figure 1a. In particular, preferred embodiment integrated circuit 100 includes the following items: CCD Controller 102 interfaced with either CCD or CMOS imager 150; preview engine block 104 to convert the data from CCD controller 102 into a format suitable for display using NTSC encoder 106 or a digital LCD interface; burst mode compression-decompression engine 108 to compress the raw image data from CCD controller 102 using a lossless (or lossy, as selected by the user) compression and to write the compressed data to external SDRAM 160 via SDRAM controller 110. This data can then be decompressed by the decompression engine under DSP 122 control, processed, and displayed or stored back to SDRAM 160. DSP subsystem block 120 (DSP 122 and iMX 124 plus Variable Length Coder 126 and buffers 128) performs all the processing of the image data in the capture mode. The data is fetched from SDRAM 160 into image buffer 128 by DSP 122 through requests to SDRAM controller 110, and DSP 122 performs all the image processing and compression required in the capture mode. The Image Extension processor (iMX) 124 acts as a dedicated accelerator to DSP 122 to increase the performance of DSP 122 for the imaging applications.

RISC microprocessor subsystem (ARM 130 plus memory 132) supports the in-camera Operating Systems (OS). Various OSes and other real-time kernels such as VxWorks, Microitron, Nucleus, and PSOS may be supported on circuit 100.

SDRAM controller block 110 acts as the main interface between SDRAM 160 and all the function blocks such as the processors (ARM 130, DSP 122), CCD controller 102, TV encoder 106, preview engine 104, etc. SDRAM controller 110 may support up to 80 MHz SDRAM timing and also provide a low overhead for continuous data accesses. It also has the ability to prioritize the

access units to support the real-time data stream of CCD data in and TV display data out.

Camera shot-to-shot delay is the time it takes for DSC engine 100 to read the data from CCD 150, process it and write it to SDRAM 160. The processing includes the image pipeline stages and also JPEG compression.

In order to support real-time preview, DSC engine 100 will set CCD 150 in "fast readout" mode, process the data, convert the data to NTSC format, and display the data on a built-in LCD screen (not shown in Figure 1) or TV monitor as the case may be.

Auto focus, auto exposure and auto white balance (the 3A functions) are performed by DSP 122 while DSC 100 is in the preview mode of operation. DSP 122 reads the image data from SDRAM 160, performs the 3A functions in real-time. The algorithms for the 3A functions are programmable.

Both interlace and progressive CCD and CMOS imagers 150 interface directly to DSC engine 100 using the built-in CCD/CMOS controller 102.

In-camera operating systems such as Microitron will be supported efficiently on ARM processor 130 in DSC engine 100. DSC engine 100 also has the capability to support capturing of a rapid sequence of images in the "burst mode" of operation. Bursts at up to 10 frames/sec of 2 Megapixel images will be supported. The duration of the burst sequence is only limited by the size of SDRAM 160 of the DSC system. Also, MPEG compression may be used for short clips. And capabilities for playback of audio-video include circular buffering.

DSC circuit 100 also includes I/O block 140 with USB core 142 for programming and interrupt processing with ARM 130.

CCD module 150 includes a CCD imager to sense the images, driver electronics and a timing generator for the necessary signals to clock the CCD, correlated double sampling and automatic gain control electronics. This CCD data is then digitized and fed into the DSC Engine 100.

SDRAM 160 may be any convenient size and speed SDRAM.

DSC systems may be even more versatile with the ability to annotate images with text/speech. The preferred embodiment programmable DSP allows

easy inclusion of a modem and/or a TCP/IP interface for direct connection to the Internet. DSCs may run complex multi-tasking operating systems to schedule the various real-time tasks.

Thus the preferred embodiments provide platforms for programmable camera functions, dual processors (ARM and DSP) plus an image coprocessor, burst mode compression/decompression engine, programmable preview engine, and integration of all camera peripherals including IrDA, USB, NTSC/PAL encoder, DACs for RGB, UART, and compact flash card/smart media card interface. Further, the platforms can provide both camera functions and digital audio playback on the same integrated circuit.

The following sections provide more detail of the functions and modules.

DSC operating modes

The preferred embodiment systems have (1) Preview mode, (2) Capture mode, (3) Playback mode, and (4) Burst mode of operation as follows.

(1) Preview mode has data flow as illustrated in Figure 2. ARM 130 sets CCD 150 into high-frame-rate readout mode (reduced vertical resolution). ARM 130 enables preview engine 104 and sets the appropriate registers for the default parameters. The raw CCD data is streamed into preview engine 104 and, after preview engine processing, is streamed into SDRAM 160. ARM 130 enables TV encoder 106 to display the preview engine output. Preview engine 104 processing (hardware) includes gain control, white balance, CFA interpolation, down-sampling, gamma correction, and RGB to YUV conversion. ARM 130 commands DSP 122 to perform auto exposure and auto white balance whenever required. DSP 122 processing includes auto exposure, auto white balance, and auto focus. ARM 130 receives new parameters for preview engine 104 and loads the preview engine hardware with these parameters. The output is full resolution CCIR 601 NTSC/PAL and real-time updating of gain, white balance, and auto focus.

(2) Capture mode has data flow as illustrated in Figure 3a. ARM 130 sets CCD 150 in "fine" readout mode, full resolution. The CCD data is read directly

into SDRAM 160 through SDRAM controller 110. ARM 130 commands DSP 122 (plus IMX 124 and VLC engine 126) perform capture processing: black clamp, fault pixel correction, shading compensation, white balancing, gamma correction, CFA interpolation, color space conversion, edge enhancement, false color suppression, 4:2:0 down-sampling, and JPEG compression. The DSP stores compressed data in the SDRAM. ARM 130 writes the compressed data to compact flash/smart media 182.

The computation is scheduled as two threads: iMX on one thread, the other units on the other thread. Figure 3b shows timing and data flow with threads related to buffers A and B.

(3) Playback mode has data flow as illustrated in Figure 4. ARM 130 reads the compressed data from CFC/Smartmeda 182 into SDRAM 160 through the SDRAM controller 110 using DMA 162. ARM commands DSP 122 to do "playback". DSP processing (DSP 122 plus IMX 124 and VLC engine 126) includes JPEG decode (bitstream parsing, IDCT, VLD, and down-sampling for aspect ratio) and store uncompressed image data in SDRAM. ARM enables TV encoder 106 to display the image on TV/LCD display. Note that also audio plus video (e.g., MPEG compressed) clips may be played back.

(4) Burst capture mode has data flow as illustrated in Figure 5, and Figure 6 shows offline data processing. ARM 130 sets CCD 150 into fine resolution mode. ARM sets up the burst compression parameters, burst length, number of frames/second, compression ratio (lossy, lossless), etc. ARM enables burst compression engine 108 to write the raw CCD data to SDRAM 160. ARM signals DSP to process each of the stored raw CCD images in the burst. Burst mode decompression engine 108 decompresses each of the burst captured images. DSP processes each of the images as in normal capture and writes the JPEG bitstream to SDRAM 160.

Burst capture mode is achieved by repeated calls to the regular playback routine with a different JPEG bitstream each time by ARM 130.

The preferred embodiment also has MPEG1 capture mode and playback mode.

Image acquisition

A DSC usually has to perform multiple processing steps before a high quality image can be stored. The first step is the image acquisition. The intensity distribution reflected from the scene is mapped by an optical system onto the imager. The preferred embodiments use CCDs, but a shift to CMOS does not alter the image processing principles. To provide a color image the imager (CCD or CMOS) has each pixel masked by a color filter (such as a deposited dye on each CCD photosite). This raw imager data is normally referred as a Color-Filtered Array (CFA). The masking pattern of the array of pixels in the CCD as well as the filter color primaries vary between different manufactures. In DSC applications, the CFA pattern that is most commonly used is an RGB Bayer pattern that consists of 2x2 cell elements which are tiled across the entire CCD-array. Figure 7a depicts a subset of this Bayer pattern in the matrix block following the CCD camera. Note that half of the pixels are sensitive to green and that the red and blue are balanced to green. Figure 7b shows a subset of the alternative complementary color CFA pattern with yellow, cyan, green, and magenta pixels.

Image pipeline

CFA data needs to undergo a significant amount of image processing before the image can be finally presented in a usable format for compression or display. All these processing stages are collectively called the "image pipeline". The preferred embodiment DSC may perform multiple processing steps before a high quality image can be stored. Most of the image pipeline processing tasks are multiply-accumulate (MAC) intensive operations, making a DSP a preferred platform. The various image pipeline processing stages are described in the following sections.

A/D converters

The A/D converter digitizing the CCD imager data may have a resolution of 10 to 12 bits. This allows for a good dynamic range in representing the input image values. Of course, higher resolution implies higher quality images but

more computations and slower processing; and lower resolution implies the converse. The A/D converter may be part of the CCD module.

Black clamp

After A/D conversion the "black" pixels do not necessarily have a 0 value due to a CCD which may still record some current (charge accumulation) at these pixel locations. In order to optimize the dynamic range of the pixel values represented by the CCD imager, the pixels representing black should have a 0 value. The black clamp function adjusts for this by subtracting an offset from each pixel value. Note that there is only one color channel per pixel at this stage of the processing.

Fault pixel interpolation

CCD-arrays may have defective (missing) pixels, especially arrays with more than 500,000 elements. The missing pixel values are filled by simple interpolation. A high order interpolation may not be necessary because an interpolation is also performed in the CFA interpolation stage. Therefore, the main reason for this preliminary interpolation step is to make the image processing regular by eliminating missing data.

Typically, the locations of the missing pixels are obtained from the CCD manufacturer. The faulty pixel locations can also be computed by the DSC engine offline. For example, during camera initialization operation, an image with the lens cap closed is captured. The faulty pixels appear as "white spots" while the rest of the image is dark. The faulty pixel locations can then be identified with a simple threshold detector and stored in memory as a bitmap.

During the normal operation of the DSC the image values at the faulty pixel locations are filled by a simple bilinear interpolation technique.

Lens distortion compensation

Due to non-linearities introduced by imperfections in lenses, the brightness of the image decreases from the center of the image to the borders of the image. The effects of these lens distortions are compensated by adjustment of the brightness of each pixel as a function fo its spatial location. The

parameters describing the lens distortions need to be measured with the final system, supported by information supplied by the lens manufacturer.

The lens adjustment can be accomplished by multiplying the pixel intensity with a constant, where the value of the constant varies with the pixel location. The adjustment needs to be done for both horizontal and vertical directions.

White balance

White balancing tries to transform the tristimulus values sensed under a certain light condition such that if displayed white appears again as white. In general the colors as captured by the camera do not appear on an output device as they were seen when capturing the scene. A couple of reasons account for that.

First, the sensitivity of the color filters over the spectral range are slightly different. If exposed with a perfect white light source (constant light spectrum) the tristimulus values sensed by the CCD are slightly different.

Second, the design of the entire CCD module and the optical system add to the imbalance of the tristimulus values.

Third, typical illuminants present while recording a scene are not constant. The illuminants have a certain "color", which is typically characterised as "color temperature" (or correlated color temperature). If an image captured under illuminant 1 is displayed under a different illuminant the color appearance changes. This causes a white area to turn a little bit red or a little bit blue.

Several different approaches for white balancing are known. Most of them multiply the red and blue channels with a factor such that the resulting tristimuls value for a white patch has identical values:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} a1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & a2 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \quad R' = G' = B' \text{ for a neutral (gray) patch}$$

However, as explained later, this approach does not provide correction for changes of the illuminant. Therefore, the white balancing implementation in preferred embodiment system corrects imbalances of the sensor module. The illumination correction is handled at a later stage in the color correction section.

Typical techniques to calculate the gain factors are

(1) equal energy

$$a1 = \sum_{(x,y)} g^2(x,y) / \sum_{(x,y)} r^2(x,y)$$

(2) gray world assumption

$$a1 = \sum_{(x,y)} g(x,y) / \sum_{(x,y)} r(x,y)$$

(3) maximum value in an image is white

$$a1 = \max_{(x,y)} g(x,y) / \max_{(x,y)} r(x,y)$$

All of them do not hold in every case. Therefore, by defining the white balancing mainly as a correction of imager module characteristics, the algorithms to obtain the correction values can be made almost scene independent.

The Figure 8 depicts the simplified realization of the preview engine, giving good results as long as the CCD sensor operates in the linear range. The white balance section below discusses a more sophisticated method.

Gamma correction

Display devices (TV monitors) used to display images and printers used to print images have a non-linear mapping between the image gray value and the actual displayed pixel intensities. Hence, in the preferred embodiment DSC Gamma correction stage compensates the CCD images to adjust them for eventual display/printing.

Gamma correction is a non-linear operation. The preferred embodiments implement the corrections as table look ups. The advantages of table look up are high speed and high flexibility. The look-up table data might even be provided by the camera manufacturer.

With 12-bit data, a full look-up table would have 4K entries, with each entry 8 to 12 bits. For a smaller look-up table, a piecewise linear approximation to the correction curves could be used. For example, the 6 most significant bits could address a 64-entry look-up table whose entries are pairs of values: a base value (8 to 12 bits) and a slope (6 bits). Then the product of the 6 least

significant bits and the slope is added to the base value to yield the final corrected value of 8 to 12 bits. Figure 9b illustrates a piecewise linear approximation curve, and Figure 9c the corresponding operations.

Note that LCD displays can be considered to be linear, making gamma compensation unnecessary. However, LCD display modules usually expect an NTSC input (which is already gamma compensated) and hence perform some "gamma uncorrection" (inverse gamma correction) to compensate for this expected gamma correction. So in the preferred embodiment DSCs using such LCD preview modules, still perform Gamma correction and then NTSC encode the signal before feeding it to the LCD module.

Gamma correction may be performed at the end of the all the stages of the image pipeline processing and just before going to the display. Alternatively, the image pipeline could perform the Gamma correction earlier in the pipeline: before the CFA interpolation stage.

CFA interpolation

Due to the use of a color-filtered array (CFA), the effective resolution of each of the color planes is reduced. At any given pixel location there is only one color pixel information (either of R, G, or B in the case of RGB color primaries). However, it is required to generate a full color resolution (R, G, and B) at each pixel in the DSC. To be able to do this, the missing pixel values (R and B at the G location, etc.) are reconstructed by interpolation from the values in a local neighborhood in the CFA interpolation. To take advantage of the DSP in this system a FIR-kernel is employed as interpolation filter. The length of the filter and the weights vary from one implementation to the other. Also the interband relationship has to be considered. Figure 10 describes the realization of the CFA interpolation in the hardwired preview engine module. It basically employs a 1D FIR kernel for horizontal followed by vertical interpolation.

The implementation in the DSP subsystem for high quality image processing is different in that it is fully programmable and able to utilize 2D filter kernels. Some background information and a proposal for an improved CFA interpolation technique is given in subsequent sections.

Color correction

Changes in the color appearance caused by differing illuminants between capture and playback/print cannot be corrected just by balancing the red, green and blue channels independently. To compensate for this, a tone (color) correction matrix maps the RGB pixel values to corrected RGB pixel values that take the illuminant into account.

The principle is as follows. Let I1 denote an NxN diagonal matrix describing the recording illuminant, S the Nx3 matrix denoting the spectral characteristics of the imager module with one column vector for each color, and R the 1xN column vector describing the reflectance of the scene. The measured tristimulus value X1 at a pixel location is given by:

$$X1^T = R^T * I1 * S$$

Denoting

$$SS = S * S^T$$

we can transform the measured tristimulus value X1 into X2, we would have been measured if the scene would have been illuminated by I2:

$$X2^T = X1^T * S^T * SS^{-1} * I1^{-1} * I2 * S$$

The 3x3 transform matrix $S^T * SS^{-1} * I1^{-1} * I2 * S$ can be calculated offline, assuming that the spectral response of the sensor can be measured. Thus it is sufficient to store a set of color correction matrices for different illuminants in the camera.

Since the subjective preferences of the color appearance changes among users, it is easily possible to include these into the color correction matrix or add a separate step to the image processing pipeline (e.g. "tone scale").

Color space conversion

After the CFA interpolation and color correction, the pixels are typically in the RGB color space. Since the compression algorithm (JPEG) is based on the YCbCr color space, a color space transformation must be carried out. Also the preferred embodiment DSC generates a NTSC signal output for display on the TV and also to feed into the LCD preview. Hence an RGB to YCbCr color space conversion needs to be carried out. This is a linear transformation and each Y, Cb, Cr value is a weighted sum of the R, G, B values at that pixel location. Figure 11a illustrates the color conversion as realized in the hardwired preview engine. The DSP (playback) implementation is similar in principle but allows a higher precision conversion:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} \alpha 1 & \alpha 2 & \alpha 3 \\ \alpha 4 & \alpha 5 & \alpha 6 \\ \alpha 7 & \alpha 8 & \alpha 9 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Edge enhancement

After CFA interpolation the images appear a little "smooth" due to the low pass filtering effect of the interpolation filters. To sharpen the images it is sufficient to operate on the Y-component only. At each pixel location we compute the edge magnitude using an edge detector, which is typically a two-dimensional FIR filter. The preferred embodiment uses a 3x3 Laplace-Operator. The edge magnitude is thresholded and scaled and before being added to the original luminance (Y) image to enhance the sharpness of the image.

The edge enhancement is a high pass filter; this high pass filter also amplifies the noise. To avoid this amplified noise, a threshold mechanism is used to only enhance those portion of the image lying on an edge. The amplitude of the amplified edge may vary. The threshold operation is necessary to reduce amplification of noise. Therefore, only those pixels get enhanced which are an element of an edge. The enhancement signal added to the luminance channel can be represented graphically as in Figure 11b; the parameters t1, t2, and the slope s1 can be chosen as seen necessary to obtain the best quality.

False color suppression

Note that the edge enhancement is only performed in the Y image. At edges the interpolated images of the color channels may not be aligned well. This causes annoying rainbow-like artifacts at sharp edges. Therefore, by suppressing the color components Cb and Cr at edges in the Y-component, these artifacts can be reduced. Depending on the output of the edge detector, the color components Cb and Cr are multiplied by a factor less than 1 on a per pixel basis to suppress the false color artifacts.

Image compression

The image compression step compresses the image, typically by about 10:1 to 15:1. The preferred embodiment DSC uses JPEG compression. This is a DCT-based image compression technique that gives good performance.

Auto Exposure

Due to the varying scene brightness, to get a good overall image quality, it is necessary to control the exposure of the CCD to maximize the dynamic range of the digitized image. The main task of exposure control is to keep the sensor operating in the linear range by controling the shutter speed, and if possible the aperture of the optical system. Since closing the iris and slowing down the shutter speed compensates each other, there exists a certain parameter range in which the exposure remains unchanged. It is obvious that this can be accomplished only to a certain extent as other constraints as capturing fast moving scenes may be desired by the user.

Besides trying to keep the sensor operating in the linear range it is desirable to maximize the dynamic range of the ADC and hence the digitized image. This is done by controlling the PGA in the AFE. The processing necessary to obtain the relevant control parameters is performed on the DSP.

Auto Focus

It is also possible to automatically adjust the lens focus in a DSC through image processing. Similar to Auto Exposure, these auto focus mechanisms operate also in a feed back loop. They perform image processing to detect the quality of lens focus and move the lens motor iteratively till the image comes

sharply into focus. Auto focus may rely on edge measurements from the edge enhancement previously described.

Playback

The preferred embodiment DSCs also provide the ability for the user to view the captured images on LCD screen on the camera or on an external TV monitor. Since the captured images are stored in SDRAM (or on compact flash memory) as JPEG bitstreams, playback mode software is also provided on the DSP. This playback mode software decodes the JPEG bitstream, scales the decoded image to the appropriate spatial resolution, and displays it on the LCD screen and/or the external TV monitor.

Down-sampling

In the preferred embodiment DSC system the image during the playback mode after decoding the JPEG data is at the resolution of the CCD sensor, e.g. 2 Megapixels (1600 x 1200). This image can even be larger depending on the resolution of the CCD sensor. However, for the display purposes, this decoded data has to be down-sampled to NTSC resolution (720 x 480) before it can be fed into the NTSC encoder. Hence, the DSC should implement a down-sampling filter at the tail end of the playback mode thereby requiring additional DSP computation.

The preferred embodiment solves this problem of additional DSP computations by a DCT-domain down-sampling scheme that is included as part of the JPEG decompression module. Note that the JPEG decompression essentially involves three stages: first an entropy decoding stage, followed by an inverse quantization stage, and finally an IDCT stage. In JPEG the IDCT is performed on a block of 8 x 8 pixels. The preferred embodiments down sample a 2 Megapixel image to NTSC resolution (a 4/8 down-sampling) in the IDCT domain by employing a 4 x 4 IDCT to the top left 4 x 4 DCT coefficients (out of a 8 x 8 DCT coefficient block) and hence effectively achieving both the IDCT and

the 4/8 down-sampling in one step. The sampling ratio can be varied between 1/8 (smallest image) to 8/8 (full resolution image).

A separable two-dimensional 4-point IDCT is applied to obtain a 4 x 4 block of image pixels from the top-left (low spatial frequency) 4 x 4 DCT coefficients. By this low-order IDCT we effectively combine anti-aliasing filtering and 8-to-4 decimation. The employed anti-aliasing filter corresponds to a simple operation of preserving only the 16 lowest frequency components in the DCT domain without scaling the preserved DCT coefficients. Though this simple filter is effective in reducing aliasing effect, the preferred embodiments may have a lowpass filter with better frequency response to further reduce aliasing. The use of other lowpass filters will lead to scaling of the preserved coefficients where the scaling factor is the location of each DCT coefficient.

Note that the DCT domain down-sampling technique does not increase the computational complexity. In fact, it reduces the computation since the JPEG decoding stages after entropy decoding does not need to deal with the whole 8 x 8 DCT coefficients except the top-left 4 x 4 coefficients. Use of other anti-aliasing filters also does not add any complexity since the coefficient scaling operation can be merged into the low-order IDCT operation. Also note that this DCT domain down-sampling idea technique can offer n/8 down-sampling ratios, n = 1, ..., 7, for other CCD sensor resolutions.

Up-Sampling

Displaying cropped images for zooming of images also uses an up-sampling scheme. The inverse approach to the down-sampling provides an elegant tool. In the first case the 8x8 DCT coefficients are (virtually) vertically and horizontally extended with zeroes to form a block of NxM coefficients (N,M >8). On this block an IDCT of size NxM is executed yielding NxM samples in the spatial domain.

Currently, most image pipeline operations are non-standardized. Having a programmable DSC engine offers the ability to upgrade the software to conform to new standards or improve image pipeline quality. Unused performance can be

dedicated to other tasks, such as human interface, voice annotation, audio recording/compression, modem, wireless communication, etc.

Figure 27 shows a preprocessing functional block diagram including CFA interpolation, white balance, color correction, tone scaling, gamma correction, conversion of RGB to YCrCb, edge enhancement, edge detection, color boost, and false color suppression in preparation of JPEG compression. The following sections describe preferred embodiments relating to CFA interpolations.

CFA interpolation with reduced aliasing

A preferred embodiment CFA interpolation for a Bayer pattern (Figure 7a) uses the high-frequency from the green channel to modify the red and blue channel interpolations to reduce the aliasing components at edges within the image by utilizing the signal of the other color channels. By this means artifacts are reduced, sharpness improved, and additional post-processing avoided. Indeed, proceed as follows.

(1) apply interpolation to green channel (any interpolation method); this yields the green plane.

(2) detect edges in the green channel (by gradient or other method).

(3) compute high-pass component of the green channel (filter with any high-pass filter).

(4) apply interpolation to the red channel (any interpolation method); this yields the red plane.

(5) add high-pass component of (3) (with a weighting factor) to red channel.

(6) apply interpolation to the blue channel (any interpolation method); this yields the blue plane.

(7) add high-pass component of (3) (with a weighting factor) to the blue channel.

So the final image consists of three color planes: the green plane from step (1), the red plane from step (5), and the blue plane from step (7). That is, for a pixel in the final image the green intensity is taken to be the value of the corresponding pixel of the green plane from step (3), the red intensity is taken to

be the value of the corresponding pixel of the modified red plane from step (5), and the blue intensity is taken to be the value of the corresponding pixel of the modified blue plane from step (7)

Theoretical analysis of the foregoing: Each CCD pixel averages the incident optical signal over the spatial extent of the pixel; thus the CCD effectively provides a low-pass filtering of the incident optical signal with a cutoff frequency the reciprocal of the pixel size. Further, the subsampling of the pixel array by the color filters on the pixels leads to aliasing in each color plane. Indeed, for red and blue the subsampling is by a factor of 2 in each direction; so the frequency spectrum folds at half the maximum frequency in each direction. Thus the red and blue baseband spectra areas are each one-quarter of the original array spectrum area (reflecting that the red and blue samplings are each one-quarter of the original array). For green the subsampling is only half as bad in that the spectrum folding is in the diagonal directions and at a distance $\sqrt{2}$ as large as for the red and blue. The green baseband spectrum is one-half the area of the original array spectrum.

Color fringing at edges is an aliasing problem. In addition, dissimilar baseband spectra lead to color fringing as well, even if no aliasing is present. Indeed, aliasing is not necessarily visible in a single color band image, but the effect becomes obvious upon combination of the three color components into one color image. The shift of the sampling grids between red, green, and blue causes a phase shift of the aliasing signal components. A one-dimensional example clarifies this: presume a one-dimensional discrete signal f(n) and two subsamplings, each by a factor of 2 but one of even-numbered samples and one of odd-numbered samples (so there is a shift of the sampling grids by one sample):

$$f_{even}(2m) = f(2m)$$
$$f_{even}(2m+1) = 0$$

$$f_{odd}(2m) = 0$$

$$f_{odd}(2m+1) \quad = f(2m+1)$$

Of course, $f(n) = f_{even}(n) + f_{odd}(n)$. Let $F(z)$ be the z-transform of $f(n)$, $F_{even}(z)$ the z-transform of $f_{even}(n)$, and $F_{odd}(z)$ the z-transform of $f_{odd}(n)$. Then noting that $F_{even}(z)$ is an even function of z (only even powers of z) and $F_{odd}(z)$ an odd function of z (only odd powers of z):

$$F_{even}(z) \quad = \{F(z) + F(-z)\}/2$$
$$F_{odd}(z) \quad = \{F(z) - F(-z)\}/2$$

The $F(-z)$ corresponds to the aliasing and appears with opposite signs; that is, a phase shift of π.

The color fringing can be reduced by a phase shift of π of the aliased components. However, this is very difficult to achieve, because the only available signal is the sum of the original signal with the aliasing signal. Therefore, the preferred embodiments have another approach.

As long as two (or more) subsampled signals (i.e., red, green, and blue) have identical characteristics (such as for a gray scale image), a perfect reconstruction of the original image can be achieved by just adding the subsampled signals. However, in CFA interpolation generally the subsampled signals stem from different color bands. Aliasing errors become visible especially at edges where the interpolated signals of the different color bands are misaligned. Therefore, the preferred embodiments counter color fringing at edges by reducing the aliasing components only at edges through utilization of other ones of the subsampled signals. This reduces artifacts, improves sharpness, and avoids additional postprocessing.

In particular, for Bayer pattern CFA the green channel has a higher cutoff frequency than that of the red and blue channels; thus the green channel has less significant aliasing. The aliasing signal to be compensated is a high-pass signal, which is now estimated as the high-pass component of the green channel; and this is added (rather than subtracted due to the phase shift due to the offset

of the red and blue subsampling grids relative to the green subsampling grid) to the red and blue channels. The high-pass green component could be multiplied by a scale factor prior to addition to the red and blue subsamplings. The signals are added while interpolating red, blue or afterwards.

CFA interpolation with inter-hue adaptation

Alternative CFA interpolation preferred embodiments first interpolate Bayer pattern greens using a 5x5 FIR filter, and then use the interpolated green to interpolate red and blue each with two steps: first interpolate diagonally to form a pattern analogous to the original green pattern (this interpolation uses a normalization by the green to estimate high frequencies), and then apply a four-nearest neighbor interpolation (again using green normalization to estimate high frequencies) to complete the red or blue plane.

More explicitly, denote the CFA value for pixel location (y,x), where y is the row number and x the column number of the array, as follows: red values R(y,x) at pixel locations (y,x) where y and x are both even integers, blue values B(y,x) where y and x are both odd integers, and green values g(y,x) elsewhere, that is, where y+x is an odd integer.

First, let G^(y,x) denote the green value at pixel location (y,x) resulting from the green plane interpolation; this is defined for all pixel locations (y,x). This interpolation can be done by various methods, including the edge preservation interpolation of the following section. Note that many interpolations do not change the original green values; that is, G^(y,x) = G(y,x) may be true for (y,x) where G was originally defined (i.e., y+x is an odd integer).

Next, define the red and blue interpolations each in two steps as illustrated in Figure 28 which is labeled for blue and uses arrows to show interpolation contributions.

First red step: R(y,x) is already defined for pixel locations (y,x) with y=2m, and x=2n with m and n integers; so first for y=2m+1 and x=2n+1, define R^(y,x):

$$\hat{R}(y,x) = \hat{G}(y,x)\{R(y-1,x-1)/\hat{G}(y-1,x-1) + R(y-1,x+1)/\hat{G}(y-1,x+1) + R(y+1,x-1)/\hat{G}(y+1,x-1) + R(y+1,x+1)/\hat{G}(y+1,x+1)\}/4$$

This interpolates the red plane to the pixels where B(y,x) was defined. (Figure 28 illustrates the analogous interpolation for blue.) Note that the this interpolation essentially averages the red values at the four corners of the 3x3 square about (y,x) with the values normalized at each location by the corresponding green values. If any of the green values are below a threshold, then omit the normalization and just average the red values.

Perform the first blue step in parallel with the first red step because the same green values are being used.

First blue step: B(y,x) is already defined for pixel locations (y,x) with y=2m+1, and x=2n+1 with m and n integers, so first for y=2m and x=2n, define B^(y,x):

$$B\hat{}(y,x) = G\hat{}(y,x)\{B(y-1,x-1)/G\hat{}(y-1,x-1) + B(y-1,x+1)/G\hat{}(y-1,x+1) + B(y+1,x-1)/G\hat{}(y+1,x-1) + B(y+1,x+1)/G\hat{}(y+1,x+1)\}/4$$

This interpolates the blue plane to the pixels where R(y,x) was defined as illustrated in the lefthand portion of Figure 28. Again, this interpolation essentially averages the blue values at the four corners of the 3x3 square about (y,x) with the values normalized at each location by the corresponding green values.

Second red step: define R^(y,x) where y+x is an odd integer (either y=2m and x=2n+1 or y=2m+1 and x=2n)

$$R\hat{}(y,x) = G\hat{}(y,x)\,[R\hat{}(y-1,x)/G\hat{}(y-1,x) + R\hat{}(y,x+1)/G\hat{}(y,x+1) + R\hat{}(y+1,x)/G\hat{}(y+1,x) + R\hat{}(y,x+1)/G\hat{}(y,x+1)]\,/4$$

This second step interpolates the red plane portion defined by the first step to the pixels where G(y,x) is defined. Again, this interpolation essentially averages the red values at four neighboring pixels of (y,x) with the values normalized at each location by the corresponding green values.

Second blue step: define for y+x an odd integer (either y=2m and x=2n+1 or y=2m+1 and x=2n)

$$B\hat{}(y,x) = G\hat{}(y,x)\{B\hat{}(y-1,x)/G\hat{}(y-1,x) + B\hat{}(y,x+1)/G\hat{}(y,x+1) + B\hat{}(y+1,x)/G\hat{}(y+1,x) + B\hat{}(y,x+1)/G\hat{}(y,x+1)\}/4$$

This second step interpolates the blue plane portion defined by the first step to the pixels where G(y,x) is defined. Again, this interpolation essentially averages

the blue values at four neighboring pixels of (y,x) with the values normalized at each location by the corresponding green values.

The final color image is defined by the three interpolated color planes: G^(y,x), R^(y,x), and B^(y,x). The particular interpolation used for G^(y,x) will be reflected in the normalizations for the two-step interpolations used for R^(y,x) and B^(y,x).

CFA interpolation with edge preservation

Alternative CFA interpolation preferred embodiments interpolate Bayer pattern greens by a (small) FIR filter plus preserve edges by a comparison of an interpolated pixel green value with the nearest-neighbor pixel green values and a replacement of the interpolated value with a neighbor value if the interpolated value is out of range. Figure 29 illustrates the green interpolation. After this green interpolation, interpolate the red and blue planes.

In particular, first at each pixel (y,x) apply the following 5x5 FIR filter to G(y,x) defined on the pixels (y,x) where x+y is odd to yield G1(y,x) defined for all (y,x):

$$1/200 \begin{vmatrix} 0 & -11 & 0 & -11 & 0 \\ -11 & 0 & 72 & 0 & -11 \\ 0 & 72 & 200 & 72 & 0 \\ -11 & 0 & 72 & 0 & -11 \\ 0 & -11 & 0 & -11 & 0 \end{vmatrix}$$

The 200 center entry just implies for (y,x) where G(y,x) is defined in the CFA, G1(y,x) = G(y,x). Note that green values are in the range of 0-255, and negative values are truncated to 0. Of course, other FIR filters could be used, but this one is simple and effective.

Next, for the (y,x) where G1(y,x) is interpolated, consider the four nearest neighbors' values G(y±1,x), G(y,x±1) and discard the largest and smallest values. Let A and B be the remaining two nearest-neighbor values with B greater than or

equal to A. Then define the final interpolated green value G^(y,x) as follows:

$$G\hat{}(y,x) = \begin{cases} A & \text{if } G1(y,x) < A \\ G1(y,x) & \text{if } A \le G1(y,x) \le B \\ B & \text{if } B < G1(y,x) \end{cases}$$

This clamps the interpolated value to midrange of the neighboring pixel values and prevents a single beyond-the-edge nearest-neighbor pixel from diluting the interpolated pixel value. Figure 29 shows the overall green interpolation.

Complete the image by red and blue interpolations. The red and blue interpolations may each be a single step interpolation, or each be a two-step interpolation as described in the foregoing section which uses the edge-preserved green values, or each be some other type of interpolation.

CFA interpolation plus noise filtering

Preferred embodiments save on line memory required for CFA interpolation followed by lowpass filtering to limit noise with an integrated approach. In particular, CFA interpolation typically contains a horizontal interpolation block and a vertical interpolation block with line memories in between as illustrated in Figure 30. The horizontal interpolation block has an input of a row of CFA signals, two toggle switches, two zero insertion subblocks, two three-tap FIR filters (coefficients 0.5, 1.0, 0.5), and two outputs: one output for each color. Each of the FIR filters just reproduces the input color values and puts the average of successive input color values in place of the inserted zeros. The zero-insertion and toggle timing of two subblocks alternate with each other. The block diagram of the horizontal interpolation block is shown in Figure 31 with a row of raw data R/G/R/G/R ... ; in this block row-interpolated Red and Green signals are output. In case the row of raw data input is B/G/B/G/B... interpolated Blue and Green signals are output.

A line (row) memory delays the data by one CFA line (row) period in order to interpolate the data in the vertical interpolation block. Figure 32 shows the four line memories and the input/output data of the memories. In the case of an input row of R/G/R/G/... raw data with m indicating the (even) row number and n the column number which increments as the row data enters, the input and output data are:

Input_A = R(m,n)

Output_A1 = Input_A = R(m,n)

Output_A2 = G(m-1,n) which was the interpolated green from the previous row of raw data, a G/B/G/B... row

Output_ A3 = R(m-2,n) which was the interpolated red from the second previous row of raw data, a R/G/R/G/... row

Input_B = G(m,n)

Output_B1 = Input_B = G(m,n)

Output_B2 = B(m-1,n) which was the interpolated blue from the previous row of raw data, a G/B/G/B/... row

Output_B3 = G(m-2,n) which was the interpolated green from the second previous row of raw data, a R/G/R/G/... row

This provides the two rows of red, R(m,n) and R(m-2,n), for vertical interpolation to create the m-1 row of red and also provides the green rows G(m,n), G(m-1,n), and G(m-2,n) which do not need vertical interpolation.

The next input row (row m+1) of G/B/G/B/... raw data leads to the following input and output data:

Input_A = G(m+1,n)

Output_A1 = Input_A = G(m+1,n)

Output_A2 = R(m,n) which was the interpolated red from the previous row of raw data, a R/G/R/G/.. row

Output_ A3 = G(m-1,n) which was the interpolated green from the second previous row of raw data, a G/B/G/B/... row

Input_B = B(m+1,n)

Output_B1 = Input_B = B(m+1,n)

Output_B2 = G(m,n) which was the interpolated green from the previous row of raw data, a R/G/R/G/... row

Output_B3 = B(m-1,n) which was the interpolated blue from the second previous row of raw data, a G/B/G/B/... row

This provides the two rows of blue, B(m+1,n) and B(m-1,n), for vertical interpolation to define the m row blue and also provides the green rows G(m+1,n),

G(m,n), and G(m-1,n) which do not need vertical interpolation.

Figure 33 shows the combinations for vertical interpolations. In particular, for row m output (row m+1 input) the combinations are (Figure 33b):

green is G(m,n)

red is R(m,n)

blue is (B(m-1,n) + B(m+1,n))/2

And for row m-1 output (row m input) the combinations are (Figure 33a):

green is G(m-1,n)

red is (R(m,n) + R(m-2,n))/2

blue is B(m-1,n)

As Figure 33 illustrates, a vertical low-pass noise filter can be applied directly to the three green outputs (G(m–2,n), G(m-1,n), and G(m,n) for row m input and G(m–1,n), G(m,n), and G(m+1,n) for row m+1 input), but red and blue cannot be vertically filtered because the four line memories of Figure 32 do not output enough lines (rows). Rather, eight line memories are needed as illustrated in Figure 34.

Figures 35a-35b illustrate the preferred embodiment combination vertical interpolation and low-pass noise filtering including green vertical noise reduction filter block A, green-noise block B, blue/red green-noise difference block C, and red/blue green-noise sum block D. The six inputs for the preferred embodiments of Figures 35a-35b are the outputs of the horizontal interpolations and four line memories of Figures 30-32 and thus the same as the inputs to the known vertical interpolation filter of Figure 34.

For an implementation of this interpolation plus noise filtering on a programmable processor the eight line memories in Figure 34 would take up twice as much processor memory space as the four line memories of Figures 30-32, and this can be significant memory space. For a large CFA such as a 2 megapixel (1920 by 1080 pixels) CCD, a line memory would be 1-2 kbytes, so the difference would be 4-8 kbytes of processor memory.

In more detail, Figure 35a illustrates the noise reduction and vertical interpolation for the case of input row m with m an even integer (raw CFA data

R/G/R/G/...) into the horizontal interpolator plus four line memories of Figure 32: the six (horizontally interpolated) inputs at the lefthand edge of Figure 35a are R(m,n), G(m-1,n), R(m-2,n), G(m,n), B(m-1,n), and G(m-2,n) (i.e., the outputs in Figure 32); and the output will be noise-reduced colors for row m-1: R"(m-1,n), G"(m-1,n), and B"(m-1,n). First, the vertical interpolation (lefthand portion of Figure 35a) averages R(m,n) and R(m-2,n) to create R(m-1,n); G(m-1,n) and B(m-1,n) already exist as inputs.

Then the noise reduction filter (block A in the righthand portion of Figure 35a) creates and outputs the vertically low-pass filtered green G"(m-1,n) as:

$$G"(m-1,n) = [G(m,n) + 2*G(m-1,n) + G(m-2,n)]/4$$

Next, block B creates Delta_G as the difference between G and G"; that is, Delta_G is the vertical high-frequency part of G:

$$Delta\_G(m-1,n) = G(m-1,n) - G"(m-1,n)$$

Because G is sampled twice as frequently as B and R in the Bayer CFA, direct high-frequency estimation of G will likely be better than that of B and R, and thus the preferred embodiment uses Delta_G to subtract for noise reduction. Note that the difference between the vertical average [G(m+1,n) - G(m-1,n)]/2 and G"(m,n) equals -Delta_G(m,n), so for R and B which are to be vertically interpolated (averaged) plus low-pass filtered, the high-frequency estimate provided by G which is to be subtracted from R and B will have opposite sign.

Thus block C subtracts Delta_G from B to create B" for row m-1 because B is not vertically interpolated for m-1:

$$B"(m-1,n) = B(m-1,n) - Delta\_G(m-1,n)$$

Essentially, the vertical high-frequency part of G is used as an estimate for the vertical high-frequency part of B, and no direct vertical low-pass filtering of B is applied.

Then block D adds Delta_G to R to create R" for row m-1 because R was vertically interpolated:

$$R"(m-1,n) = R(m-1,n) + Delta\_G(m-1,n)$$

Again, the vertical high-frequency part of G is used in lieu of the high-frequency part of R, and because an vertical averaging creates R(m-1,n), the opposite sign

of Delta_G is used to subtract the high-frequency estimate.

Thus the noise-reduced filtered three color output row m-1 are the foregoing G"(m-1,n), R"(m-1,n), and B"(m-1,n).

Similarly, for output row m from input row m+1 (again with m an even integer) and raw CFA data G/B/G/B/... the six (horizontally interpolated) inputs are G(m+1,n), R(m,n), G(m−1,n), B(m+1,n), G(m,n), and B(m-1,n), and the output will be noise-reduced colors for row m: R"(m,n), G"(m,n), and B"(m,n). The vertical interpolation (lefthand portion of Figure 35b) averages B(m+1,n) and B(m-1,n) to create B(m,n); G(m,n) and R(m,n) already exist as inputs. Then the noise reduction filter (righthand portion of Figure 35b) block A again creates vertically low-pass filtered green G"(m,n) as:

$$G"(m,n) = \{G(m+1,n) + 2*G(m,n) + G(m-1,n)\}/4$$

Next, block B again creates the vertical high-frequency portion of G, called Delta_G, as the difference between G and G":

$$Delta\_G(m,n) = G(m,n) - G"(m,n)$$

Then block C again subtracts Delta_G but from R (rather than B as for row m-1 outputs) to create R":

$$R"(m,n) = R(m,n) - Delta\_G(m,n)$$

Thus the high-frequency part of G is again used as an estimate for the noisy part of R, and no direct noise filtering of R is applied, but for row m the Delta_G is subtracted rather than added as for row m-1. Indeed, for R even rows have Delta_G subtracted and odd rows have Delta_G added because the odd rows have R defined as a vertical average.

Lastly, block D adds Delta_G to B to create B":

$$B"(m,n) = B(m,n) + Delta\_G(m,n)$$

Thus as with R, the Delta_G vertical high-frequency estimate is row-by-row alternately added to and subtracted from B instead of a direct vertical low-pass filtering of B. Note that for a given row the Delta_G terms for R and B have opposite signs because one of R and B will be an average of preceding and succeeding rows.

In short, the preferred embodiments are able to emulate the CFA

horizontal interpolation, vertical interpolation, and low-pass filtering with only four line memories by using a high-frequency estimate based on G.

Figures 36a-36b and 37a-37b illustrate an alternative embodiment in which the vertical low-pass filtering of G differs from the 1/4, 1/2, 1/4 weighting of the preferred embodiments of Figures 35a-35b.

CFA interpolation for complementary color CCD

Preferred embodiment CFA interpolations for a complementary color pattern CFA (illustrated in Figure 7b) combine a simple interpolation followed by an image quality enhancement by detection and adjustment for color imbalance. In particular, presume initial interpolation as defined at each pixel all four complementary color values, and denote the color values as Ye (yellow), Cy (cyan), Mg (magneta), and G (green).

First, at each pixel compute an imbalance factor μ:

$$\mu = Ye + Cy - 2*G - Mg$$

This imbalance factor represents the difference between ideal and actual pixel color values. Indeed, the definitions of the complementary color values in terms of red value (R), green value (G), and blue value (B) are Ye = R + G, Cy = G + B, and Mg = B + G. Hence, the following relation always holds for a pixel's color values:

$$Ye + Cy = 2*G + Mg$$

Thus the imbalance factor μ ideally vanishes. When an edge is near a pixel, imbalance can arise due to the spatial difference of each of the four color samples in the CFA. The preferred embodiments detect the imbalance and adjust by modifying each color value:

$$Ye' = Ye - \mu/4$$

$$Cy' = Cy - \mu/4$$

$$Mg' = Mg + \mu/4$$

$$G' = G + \mu/8$$

Then these modified complementary colors are used to form the final image.

Figure 38 illustrates the overall flow for the enhancement using the imbalance factor. Of course, scale factors other than -1/4, -1/4, 1/4, and 1/8

could be applied to the imbalance factor provided that Ye' + Cy' = 2*G' + Mg'.

White balance

The term "white balancing" is typically used to describe algorithms, which correct the white point of the camera with respect to the light source under which the camera currently operates. Since the estimation of the true light spectrum is very difficult, the aim of most approaches is to correct the output of the red and blue channel (assuming CCDs based on the RGB color filters), such that for a gray object the pixel intensities for all color channels are almost identical. The most common technique basically calculates the average energy or simply the mean for each channel. The calculation of averages may be carried out in N local windows W_j, j = 1, 2, ..., N, as for red:

$$R_j = \Sigma_{k \in W_j} r(k)$$

with r(k) denoting the digital signal for the red channel. Similar averages B_j and G_j are calculated for the blue and green color channels. The imbalance between the channels, given by the green-to-red and green-to-blue ratios

$$WBR = \Sigma_j G_j / \Sigma_j R_j$$

$$WBB = \Sigma_j G_j / \Sigma_j B_j$$

are used as correction multiplier for the red and blue channels, respectively

$$r'(k) = WBR\ r(k)$$

$$b'(k) = WBB\ b(k)$$

There exist many different flavors of this approach, which all calculate intensity-independent multiplication factors WBR and WBB.

This approach works only if several assumptions are valid. First, it is assumed that the sensor responses are well aligned over the input intensity range; in other words, the green response curve equals the red (blue) response curve multiplied by a factor. Looking at sensor (CCD) characteristics indicates that this assumption does not hold. For high light intensities, the sensor saturates; while at very low light intensities, the sensor response (especially for the blue channel) is very small. Furthermore, non-linearities of the sensor, as well as imbalances of the color channels related to the sensor response and the light source, are handled simultaneously. Resulting artifacts include magenta

colors in very bright areas, where the "color" should turn white, or wrong colors in dark areas.

The pixel intensity at the sensor output, e.g. for the red color channel, can be modeled as

$$r(k) = \int I(\lambda)\beta(k,\lambda)f_R(\lambda)\alpha(I,\lambda)d\lambda$$

where λ denotes the wavelength, $I(\lambda)$ the spectrum of the light source, $\beta(x,\lambda)$ the reflectance of the object under observation, $f_R(\lambda)$ the spectral sensitivity of the red color filter covering the CCD pixels, and $\alpha(I,\lambda)$ the intensity- and wavelength-dependent efficiency of the CCD in converting photons into electrons.

Regarding only the spectral response curves of the color filters $f_R(\lambda)$ (and also $f_G(\lambda)$ and $f_B(\lambda)$) of a typical CCD sensor, the output signals differ:

$$WBR = \int f_G(\lambda)d\lambda \,/\, \int f_R(\lambda)d\lambda = 1.09$$

$$WBB = \int f_G(\lambda)d\lambda \,/\, \int f_B(\lambda)d\lambda = 1.34$$

The values are obtained using the response of a typical CCD and assuming perfect white light source (the spectrum $I(\lambda)$ is flat), a perfectly white object (the spectrum of the reflected light is identical to the spectrum of the illuminating light which means $\beta(k,\lambda) = 1$), and neglecting $\alpha(I,\lambda)$ (no wavelength dependent quantum efficiency). Especially the blue channel shows a smaller response than green or red at the same intensity. The non-linear quantum efficiency of the sensor is another effect. A typical s-shaped sensor response over the input intensity is shown in Figure 39a. Furthermore, the sensor response in each channel depends on spectrum of the light source.

Thus, preferred embodiment white balancing takes into account the misalignment as well as the non-linearity. Typical light sources are not flat over the visible spectrum but tend to have a higher energy in certain spectral bands. This effect influences the observed sensor response; ideally it should be corrected by white point compensation, which may be based on a correction matrix. An independent balancing of the channels cannot handle this effect as previously outlined. For ease of mathematical description, approximate the s-shaped response curve in Figure 39a by piecewise linear segments. Three

segments separate the light conditions into three categories: very low intensity, normal intensity, and very bright light. Figure 39b shows the effect of applying a single multiplier. With respect to the green signal, the amplification of the blue signal is too small in low light conditions, whereas in very bright conditions the multiplier is too large. Reducing the factor leaves an offset between the components, visible as wrong colors. Therefore, the correction terms for aligning all three response curves must look different and reflect the sensor characteristics.

The preferred embodiment white balancing splits into two separate schemes, one accounts for imager dependent adjustments, while the other one is related to light sources.

Without any restrictions on generality, the s-shape response curve is approximated in the following by three piecewise linear segments. More segments increase the accuracy but do not change the basic concept. For the first region (very low intensity) and the blue channel, the model reads with s the response and x the input intensity:

$$s_{B,1} = a_{B,1}x$$

Modeling the second region requires a multiplier and an offset

$$s_{B,2} = a_{B,2}\,x + b_{B,2}$$

The offset term is determined by the constraint that the response curve needs to be contiguous at the transition point x_1 from region 1 to region 2:

$$s_{B,1}(x_1) = s_{B,2}(x_1)$$

$$\text{so } b_{B,2} = (a_{B,1} - a_{B,2})x_1$$

The parameters for the linear model of region 3

$$s_{B,3} = a_{B,3}\,x + b_{B,3}$$

are completely determined because the maximum output has to be identical to the maximum input x_{max}, and the the response curve needs to be contiguous at the joint point x_2:

$$x_{max} = a_{B,3}\,x_{max} + b_{B,3}$$

$$s_{B,2}(x_2) = s_{B,3}(x_2)$$

$$a_{B,3} = (s_{B,2}(x_2) - x_{max})/(x_2 - x_{max})$$

$$b_{B,3} = (1 - a_{B,3})x_{max}$$

Thus the parameters to specify the approximation of the response curve for each color component are a_1, a_2, x_1, and x_2. x_{max} is not a free parameter, because it is specified by the bit resolution of the input signal.

The preferred embodiment white balancing now applies different multipliers for each region. For continuous transition from one region to the next, an additional offset is required. Although the number of regions is arbitrary, without loss of generality only three regions are considered in the following equations. The correction term for blue with respect to green for region 1 has to be:

$$WBB_1 = a_{G,1} / a_{B,1} \approx G_1/B_1$$

where window 1 (for G_1 and B_1) has pixels with intensities in region 1.

Thus, an input intensity value lying in region 1 gets the corrected output

$$b'(k) = WBB_1\, b(k)$$

Based on the balancing multiplier for region 2

$$WBB_2 = a_{G,2} / a_{B,2} \approx G_2/B_2$$

the white balancing must consider an additional offset for values in region 2

$$b'(k) = WBB_2\, b(k) + WBOB_2$$

with

$$WBOB_2 = (WBB_1 - WBB_2)\, x_1$$

For the third region the calculation is basically the same, except that no explicit WBB_3 can be specified, but the amplification is determined by the maximum value x_{max}.

$$b'(k) = WBB_3\, b(k) + WBOB_3$$

with

$$WBB_3 = (x_{max} - (WBB_2 x_2 + WBOB_2))/(x_{max} - x_2)$$

$$WBOB_3 = (1 - a_{B,3})\, x_{max}$$

For an implementation, the system must determine appropriate white balancing multipliers WBB_i for N-1 regions. Based on these values, the remaining offset values WBOB and the multiplier for the last regions are calculated. The locations of the transition points are specified a priori. The white balancing itself selects the

region based on the intensity value of the input pixel and applies the appropriate gain and offset to that value

$$b'(k) = \begin{cases} WBB_1 * b(k) & b(k) \le x_1 \\ WBB_2 * b(k) + WBOB_2 & x_1 < b(k) \le x_2 \\ WBB_3 * b(k) + WBOB_3 & x_2 < b(k) \end{cases}$$

Plus a similar multiplier for the red channel.

The total dynamic range of the CCD output signal is independent of aperture, and shutter, since they affect the number of photons captured in the CCD. An analog gain however, or any digital gain prior to processing shifts the signal and should be avoided. In case a gain (digital) α needs to be applied, this gain can be included into the white balancing method. A gain maps the maximum input value x_{max} to the output value $\alpha * x_{max}$

The scaled response curves behave identical to the non-scaled one, meaning that the scaled signal saturates at $\alpha * x_{max}$. Substituting

$$WBB_1 := \alpha * WBB_1$$

$$WBB_2 := \alpha * WBB_2$$

In that way the equation in the previous section remain unchanged, except

$$WBOB_3 = (\alpha - a_{B,3}) x_{max}$$

After linearization the signal can undergo an adjustment reflecting the light source. This is also known as white point adjustment. Here the input signal is transformed such that it looks like as if it has been captured under a different light source. For example, an image has been captured in bright sunlight (D65), but the color characteristics should be as if it has been captured under indoor conditions (D_{50} tungsten).

$$[R,G,B]D_{65}^T = I_{D65}^T * \beta * [f_R, f_G, f_B]^T$$

$$[R,G,B]D_{50}^T = I_{D50}^T * \beta * [f_R, f_G, f_B]^T$$

Here, I_{Dxx} denotes a vector sampling the light spectrum, β is a diagonal matrix describing the reflectance of the objects, and f_R, f_G, and f_B denote the spectral response of the CCD light filters. Based on these equations a 3x3 transformation matrix can be calculated relating the signal under D65 to D50:

$$[R,G,B]D_{50}^T = I_{D50}^T * I_{D65}^{-T} * [R,G,B]D_{65}^T$$

The 3x3 transformation matrix

$$M_D = I_{D50}^{T} * I_{D65}^{-T}$$

can be calculated offline.

In real systems it is almost impossible to determine averages for the different response regions. Therefore a simple solution is to calculate overall values as in the foregoing ratio of integrals, and modify them with fixed values based on predetermined sensor measurements

$$WBB_1 = \alpha_1 * WBB$$

$$WBB_2 = \alpha_2 * WBB$$

And similarly for WBR.

The transition points can be fixed in advance, too. There is just one exception for the transition point x_2. In rare situations the WBR-value may be so large that it exceeds the maximum output value at the transition point x_2. In that situation, either the WBR needs to be decreased or the transition point is reduced. The diagram in Figure 40 shows an example of the effectiveness of this technique. The red components is adjusted with respect to the green component. Using a single multiplier exceeds the green signal in bright areas, and is less effective in low light areas, whereas the segmented white balancing matches the green curve for all intensities.

Resizing preferred embodiments

Frequently images captured in one size (e.g., 320 x 240 pixels) have to be converted to another size (e.g., about 288 x 216) to match various storage or input/output formats. In general this requires a fractional up-sampling or down-sampling by a rational factor, N/M; for example, a resizing from 320 x 240 to 288 x 216 would be a 9/10 resizing. Theoretically, resizing amounts to cascaded interpolation by N, anti-aliasing filter, and decimation by M. In practice the resizing may be achieved with an M-phase, K-tap filtering plus selection of N outputs per M inputs.

For example, preliminarily consider a resizing by a ratio of 63/64 using a 3-tap filter as illustrated in Figure 41a in which the top horizontal line represents pixel

inputs and the horizontal length-three braces represent the 3-tap filter kernel applied to the indicated three inputs and producing the indicated outputs. Indeed, presume the filter kernel is a continuous function f(t) with support of length 3-1/63 so that at most three inputs can be involved; see Figure 41b. Note the slight shifting to the right of successive braces in Figure 41a: this represents the resizing from 64 inputs down to 63 outputs because the center of the filter kernel (and thus the non-rounded-off output position) must increment 1+1/63 (=64/63) pixel positions for each output in order for the 63 outputs to match the 64 inputs. Output[0] (represented by the farthest left brace in Figure 41a) is centered at the position of input, and the non-rounded-off output position j, denoted outp_pos[j], thus equals 1 + j*64/63.

The filter kernel is represented as a symmetrical continuous function f(t) centered at time 0. Output[0] for example, needs three kernel values: f(-1), f(0), and f(1). Each output point is computed as the inner product of three kernel coefficient values with three input pixel values. The center input point for the output[j] is positioned at round(outp_pos[j]) where round() is the round off function. The other two input points are offset from this center point by ±1. The center filter kernel coefficient value is f(round(outp_pos[j]) - outp_pos[j]) and the other are f() at the ±1 offsets of this center value point. Thus the following table shows the output position, coefficient kernel values, and input points needed for each output:

output j	outp_pos	center coeff position	input points
0	1	0	0,1,2
1	2 1/63	-1/63	1,2,3
2	3 2/63	-2/63	2,3,4
...	...	...	...
31	32 31/63	-31/63	31,32,33
32	33 32/63	31/63	33,34,35
33	34 33/63	30/63	34,35,36
...	...	...	...
61	62 61/63	2/63	62,63,64
62	63 62/63	1/63	63,64,65
63	65	0	64,65,66
...	...	...	...

The table shows the desired coefficient position as well as the inputs involved in each output. Note the j=63 case is similar to the j=0 case in that the kernel center aligns with the input, but with the output position and input indices shifted by 64. Notice that at j=32 there is a change in the input pattern: for j≤31, output[j] uses input j, j+1, and j+2; whereas for j≥32, output[j] uses inputs j+1, j+2, and j+3.

The preferred embodiments partition the filtering computations for resizing a two-dimensional array (image) between iMX 124 and DSP 122 and limit memory use as follows. First iMX 124 performs the 3-tap row filtering with 64 banks of coefficients and then 3-tap column filtering with 64 banks of coefficients. First consider the row filtering. 3-tap row filtering on iMX 124 has the input/output relationship:

iMX output j	input points
0	0,1,2
1	1,2,3
2	2,3,4
...	...
31	31,32,33
32	32,33,34
33	33,34,35
...	...
61	61,62,63
62	62,63,64
63	63,64,65
64	64,65,66
...	...

Comparing this table with the prior 63/64 resizing table shows that the only difference is the iMX produces one extra point, namely, IPP_output[32]. Thus the preferred embodiments produce the 64 output points with iMX 124, and then use DSP 122 to pick the 63 valid points:

output[j] = IPP_output[j] for j = 1,2,...31
IPP_outpt[j+1] for j = 32,33, ..., 62

In general, N/M resizing when N/M is less than 1 involves deleting M-N outputs of every M outputs. Thus the preferred embodiments generally perform the filter

operations on the M input points in an accelerator such as the iMX and then use a processor such as the DSP to discard the unneeded outputs. (iMX can also handle larger-than-unity resizing up to N/M = 3.)

iMX can produce 8 outputs of 3-tap row filter in 3 cycles. Basically, 8 adjacent outputs are computed in parallel using the 8 MAC units. At time 0, pull out input points 0,1,2,3,...7, multiply with appropriate coefficients (each can be different), and accumulate into 8 accumulators. At time 1 pull out input points 1,2,...8, do the same, and at time 2, pull out input points 2,3,...9, accumulate the products, and write out 8 outputs, j=0,1,...7. Next, shift over 8 input points to compute j=8,9,...15.

For the vertical direction, iMX computes 8 outputs in parallel as well. These are 8 horizontally adjacent output points, and every fetch of input array also bundles 8 horizontally adjacent output points. Therefore, all 8 MAC units share the same coefficient values for each cycle. For vertical direction there is less data reuse in iMX, so input/output memory conflicts slow down the computation to 4 cycles/8 outputs. Total filtering time is 7 cycles/8 outputs, or 7/8 cycle per output. Input data is of size 320x240x3. Thus, the filtering of iMX takes 320x240x3.7/8 201,600 cycles, or 1.7 msec with iMX running at 120 MHz.

After filtering, DSP picks correct outputs. Basically, one row out of every 64 rows and one column out of every 64 columns should be discarded. A DSP assembly loop moves the valid iMX output points to a separate output area. iMX and DSP may run in parallel if there is sufficient local memory for both. An entire input image likely is too large to fit into local memory; even the natural choice, 63x63 output points, may be too large. In such a case partition the image, such as 63 wide x 16 tall, and deal with extra bookkeeping in the vertical direction. With just 3x64 = 192 coefficients, it would be economical to pre-compute and store them. DSP should keep track of the phase of each processing block, and point iMX to the correct starting address of coefficients. If the colors are interleaved, this allows interleaved filtering as well. iMX deals with strides in getting input points. The following table shows interleaved 3-tap filtering.

j	input points
0	0,3,6

1	1,4,7
2	2,5,8
...	...

However, interleaving consumes three times more memory for the same output block size for each color. Thus it si possible to partition the task into smaller size, such as 63x5 on each color plane, and eal with extra overhead in the vertical direction. If the color format is not 4;4:4 (say, 4:2:2), and input is color-interleaved, the DSP will need to spend some additional time separating color planes.

Performing resizing totally in DSP 122 is time-consuming if implemented with straightforward fractional addressing. The preferred embodiments streamline the computation by requiring filter coefficients to be reordered and padded with dummy words. iMX 124 performs the main processing concurrently with DSP 122 computing the coefficients. This efficiently realizes high throughput resizing.

In more detail, the preferred embodiments perform an N/M resizing of an image by using iMX 124 to perform M-phase, K-tap filtering (which produces redundant output points) and DSP 122 to select the correct output points. Further, DSP 122 computes needed coefficients from a fewer-subsample coefficient template to reduce memory usage to 8*K; otherwise memory usage up to 2*M*K coefficient words would be needed. DSP 122 can compute the rounded position for the coefficients, and build up the coefficient memory for iMX 124.

For processing wide and short blocks of pixels (i.e., 16 x 64) the horizontal direction requires more computation in that horizontal coefficients are updated more often than vertical coefficients. However, the coefficients constructed by DSP 122 can be reused many times within the short block, so the load on DSP 122 should not be excessive.

In particular, preferred embodiments proceed with the following steps which are illustrated in Figures 42a-42e for a 3-tap filter and a 10-to-9 resizing (e.g., resizing from 320x240 to 288x216 in 30 frames/sec)(presume 4:4:4 interleaved, for 4:2:2 or 4:1;1 do subsampling after resizing):

1. select input/output pattern: every 10 inputs leads to 9 outputs as per Figure 42a.

2. draw coefficient pattern for a processing unit, one color first. Arrows in Figure 42b indicate which input points are used: connected arrows form the same output point, and gray (open head) arrows indicate zero coefficients. Thus three input points determine the first output point, only two input points determine each of the next eight output points, and then a tenth ignored output (no non-zero input points); and this repeats every ten. This pattern suggests use of a polyphase 3-tap filter, and drop the last output in every group of 10 outputs.

3. consider interleaved input/output. See Figure 42c which shows a set of three groups of ten input points interleaved so that the three input points determining the first output point from the original first group of ten input points are now at locations 1, 4, and 7; the three input points determining the first output point from the original second group of ten input points are now at locations 2, 5, and 8; and the three input points determining the first output point from the original third group of ten input points are now at locations 3, 6, and 9; and so forth. This interleave implies that sets of three adjacent output points use all different input points and do not require simultaneous memory accesses.

4. Consider 8-way parallelism and iMX, add more dummy outputs if necessary. See Figure 42d which shows the output points partitioned into four groups of 8 for parallel computations.

5. Compute coefficients and order as grouped. iMX will process one group at a time, using coefficient order from left-to-right, then up-to-down, then next group. Coefficients need to be arranged to the same order. If the iMX coefficient memory and the flash memory can accommodate all these coefficients, these coefficients can be included in the DSP code as constant data, and this step is done once in the software development. If the iMX coefficient memory can hold these coefficients all the time, but these take up too much room in the flash memory, this step can be performed once during system initialization. Likely the SDRAM can hold all these coefficients, but iMX coefficient memory cannot hold them all the time. this step should be performed once in the system initialization, an the coefficient image should be stored in SDRAM. When needed, these coefficients are swapped in from the SDRAM. If it is not desirable to store all these coefficients at any time,

especially when M is very large (100+), compute needed "window" of coefficients with DSP concurrently with iMX processing. Just make sure the iMX coefficient memory can hold the necessary coefficients for a computation block.

6. Start computation on iMX. In this case, it takes about 12 cycles in the inner loop to produce the 27 valid output points. Each iMX command can produce a 2-D output block, so producing 16x27 output points will take about 10 + 16*12 =202 cycles.

7. When iMX is done, have DSP pick the correct output points. In this example, 276 points are picked out of every group of 32 output points. This task will be easier to code if the width of output matches or is a multiple of 3*M. DSP only has to touch each valid outaput once, so the loading of the DSP should not be significant.

In vertical resizing, iMX works in SIMD mode. Every group of 8 adjacent data input are processed in parallel. Coefficient are used one value per cycle, and this value should apply to all color components. Even if resizing factors are the same for horizontal and vertical, how iMX uses coefficients is different, so there needs to be a separate vertical resizing coefficient storage (which takes 1/3 of horizontal coefficients). See Figure 42e. Again, there is the option to keep all vertical coefficients in iMX, swap in and out, or have DSP compute on the fly. DSP may need to pick valid output rows after iMX completes processing.

Tone-scaling preferred embodiments

Tone-scaling operates on the dynamic range of the luminance signal (or the color signals) of an image to make details more clear. For example, a picture taken against the light or in a very bright environment typically has high brightness levels. Tone-scaling commonly relies on luminance (or color) histogram equalization as illustrated in block form by Figure 43. Indeed, converter block 430 converts the input luminance levels (in the range 0 to 255 for 8-bit or 0 to 4095 for 12-bit) to output luminance levels in the same range using a look-up table. The look-up table consists of the pairs that are the input level and the corresponding output level with the output levels calculated in histogram equalization block 432 as follows. First, find the cumulative distribution function of the input luminance levels of the image to

which the tone-scaling will apply; that is, find F(r) such that F(r) = (the number of pixels with level ≤ r)/(total number of pixels in the image). Next, create the look-up table function T(r) through multiplication of F(r) by the maximum pixel level and round-off to the nearest integer. Then the look-up table is just the pairs of levels (r,s) where s = T(r). Figure 45 illustrates T(r) for an under-developed image (the majority of pixels have a low level as reflected by the large slope of T(r) for small r) in which fine details in dark parts are difficult to perceive. Also as Figure 45 shows for this under-developed image, the tone-scaling converts the level r = 500 to s = 2000; and thus in the tone-scaled image the differences of the luminance levels will be emphasized for the low levels and de-emphasized for the high levels. Thus the tone-scaling enhances detail in dark portions.

However, the tone-scaled image may look unnatural in that the colors are too clear, as if the tone-scaled image were painted in oil paints. Thus this tone-scaling is sometimes too strong for consumer use because of the unnatural character even if the fine details are clearer; although other applications such as medical and night vision demand the fine detail despite unnaturalness.

The preferred embodiments provide tone-scaling by using a linear combination of the histogram equalization function T(r) and the original image level r. That is, for a parameter α with $0 \leq \alpha \leq 1$ define a tone-scaling function by

$$s = \text{Round}(\alpha T(r) + (1-\alpha) r)$$

where T(r) is as previously described except that the round off to the nearest integer is not needed in the definition of T(r) because of the subsequent multiplication by α plus addition of $(1-\alpha)r$ and round off. Figure 45 illustrates the preferred embodiment for $\alpha = 0.3$ between the curve s = T(r) and the identity line s = r.

Figure 44 shows preferred embodiment tone-scaling in functional block form: again define a histogram equalization function T() for the luminance (or color) levels in block 442, and then define the rounded-off linear combination with weight α of T() and the identity in block 444 to yield the final look-up table for the tone-scaling in converter 440. When the weight α equals 0, then there is no tone-scaling and a natural look, but when the weight α equals 1, the tone-scaling is with

T() and fine details are enhanced. The value of weight α can be selected according to the application. All of the computations are programmable.

Implementation details

Preferred embodiment hardware structures supporting the foregoing functions include the following.

SDRAM Controller

SDRAM controller block 110 acts as the main interface between SDRAM 160 and all the function blocks such as processors (ARM 130, DSP 122), CCD controller 102, TV encoder 106, preview engine 104, etc. It supports up to 80 MHz SDRAM timing. It also provides low overhead for continuous data accesses. It also has the ability to prioritize the access units to support the real-time data stream of CCD data in and TV display data out. It also provides power down control for external SDRAM . DSP 122 can inhibit CKE signal of SDRAM 160 during no data access.

SDRAM controller block 110 supports 16/64/128/256 MB SDRAMs, 32-bit width or 2 x 16-bit width SDRAMs, maximum 80 MHz (e.g., 10-80 MHz) operation, availability of word, half-word, or byte access (ARM), commands: mode setting, power down and self refresh, programmable refresh interval, 2 or 3 CAS latency can be selectable, 2 Chip Select Output (maximum SDRAM size is 1G bit), authorizes and manages DMA transfers, manages the data flow between processors SDRAM, CCD data buffer to SDRAM, preview engine to SDRAM, burst compression to/from SDRAM, video encoder from SDRAM, OSD from SDRAM, ARM to/from SDRAM, DSP image buffer to/from SDRAM. Figure 12a shows the data flow managed by the SDRAM controller. The signals and priorities are:

Signal Name	Signal Description
Clk	SDRAM clock (10-80 MHz)
Req	Data read/write request signal
req_en	Request enable (acknowledge) signal from SDRAM Controller When the peripheral modules require a data IN/OUT, the req signal shall be asserted and when the req_en signal is asserted, the req signal shall be negated
Address	Start address of read or write CCDC, PREVIEW, BURSTC, ENC, OSD, DSP: 22-bit width ARM: 25-bit width
Odata	output data to SDRAM (32-bit)
Idata	Input data from SDRAM (32-bit)

Rw	Read or Write signal 0: Write / 1: Read
Dten	Data write enable signal for DSP IF
Ds	Bus Select (4-bit) for ARM IF

The Priority list of access units is as follows.

Priority	Access Unit
1(highest)	ENC out
2	CCD in
3	OSD out
4	PRVW in
5	BURST in
6	DSP I/O
7	ARM I/O

Preview engine

Figure 14 is a block diagram of preferred embodiment preview engine 104 which provides image data with YCbCr in 4:2:2 format from CCD raw data from CCD-controller 102 and has the following main functions.

_ Available for both RGB CCDs and complementary (YeCyMgG) CCDs (Figures 7a-7b show these CCD patterns)

_ Digital gain adjustment

_ White balance

_ Vertical and horizontal noise filter

_ RGB gain adjustment for complementary CCDs

_ Independent gamma correction for RGB colors

_ YCbCr-4:2:2 formatted data output

Sync module 1402 generates control signals for other modules such as a sync signal for a starting point of an image and an enable signal for down sampling. In this module, no image processing is executed. White balance module 1404 executes digital gain adjustment and white balance for CCD raw data. CFA interpolation module 1406 has many important sub-modules such as a horizontal noise filter, a horizontal interpolation, a vertical noise filter, a vertical interpolation, a down sampling, etc. This module outputs RGB formatted data irrespective of CCD mode (RGB CCD or complementary CCD). RGB gain

modules 1408 for complementary CCD allow adjustment to white balance by RGB color format for complementary CCD. Gamma correction modules 1410 execute gamma correction with an approximated gamma curve having 4 linear segments. This module exists for each color to permit the independent adjustment to RGB. RGB2YCbCr conversion module 1412 converts RGB formatted data into YCbCr formatted data and adjusts offsets of Cb and Cr. 4:2:2 conversion module 1414 converts YCbCr-4:4:4 formatted data into 4:2:2 format and outputs them on a 32-bit data bus. SDRAM interface module 1416 communicates with SDRAM controller 110 (Figure 1b) and requests it to store YCbCr-4:2:2 formatted image data.

The following describes the modules.

White balance module 1404 executes digital gain adjustment and white balance for CCD raw data. Digital gain adjusts for total brightness of the image and white balance adjusts the ratio of colors existing in a CFA pattern.

Figure 8 is a block diagram of white balance module 1404. There are two multipliers for the two gain adjustments and clip circuits to reduce the size of circuits. A gain value for digital gain named PVGAIN in this figure uses data in a PVGAIN register, and white balance is selected automatically by setting the CFA pattern register.

CFA interpolation module 1406 include both sub-modules for horizontal and vertical interpolation and for horizontal and vertical noise filtering, down sampling, color adjustment and complementary color to RGB color conversion. Figure 10a is a block diagram of CFA interpolation module 1406. Horizontal noise filter sub-module 1002 executes a three-tap low pass filter horizontal filter; see Figure 10b.

Horizontal interpolation filter sub-module 1004prepares two types of filters and interpolates horizontally using one of them. The outputs signal "L" and "R" means a left data and a right data on the line. For example, a processed line starts the following CFA pattern, GBGBGBGBGB ..., the output signal "L" is G and "R" is B. Therefore, these two outputs change the colors each line. Horizontal down-sampling sub-module 1006 outputs only data on valid pixels

based on register settings of horizontal decimation pattern. Vertical interpolation sub-module 1008 processes a three-tap vertical interpolation filter using two line-memories 1010 outside the preview engine module and outputs data of all colors existing in the CFA pattern. And this sub-module also executes a vertical noise filter. Color selection sub-module 1012 extracts data by each color in the CFA pattern and outputs RGB color formatted data in RGB CCD mode or complementary color formatted data in complementary CCD mode. In this figure, "g" signal is temporal data regarding G and used for recalculating R and B in the next color adjustment sub-module 1014. The color formatted data is processed color adjustment in color adjustment sub-module 1014 and the processing is different depending on CCD mode. This image processing from vertical interpolation sub-module 1008 to color adjustment sub-module 1014 has a strong correlation depending on CCD mode and vertical interpolation mode. Therefore, the processing should be considered as a sequence of vertical interpolation processing as described below. Comp2RGB conversion sub-module 1016 converts complementary color format into RGB color format in complementary CCD mode. In RGB CCD mode, the data bypass this sub-module.

The following sections describe these sub-modules.

Horizontal noise filter 1002 executes three-tap horizontal low pass filter and can reduce random noise effectively. Actually, when the center of data is set to X_0, the following calculation is executed depending on the CFA pattern and its processed line.

$$X_0 = \begin{cases} (X_{-2} + 2X_0 + X_2)/4 & \text{(two colors in processed line)} \\ (X_{-1} + 2X_0 + X_1)/4 & \text{(one color in processed line)} \end{cases}$$

An on/off switching of this filter can be controlled by a register setting.

Figure 10b is a block diagram of horizontal noise filter sub-module 1002. The two types of filter are implemented by using two adders and a switch named "three_taps_sw" in this figure. If there is one color in the processed line, the switch is set to on (High in the figure). This switch is automatically controlled depending on a register setting of the CFA pattern and a position of the line in

the processed image. Before the output, noise-filtered data or bypassed data is selected by a register setting.

In horizontal interpolation sub-module 1004, there are two modes of filtering and the data from horizontal noise filter 1002 is interpolated horizontally by either a two-tap or five-tap interpolation filter. The two-tap filter utilizes the average the two data at the adjacent pixels on the left and right to interpolate the center data. This mode is called "simple mode". The five-tap horizontal interpolation filter utilizes the information of another color on the processed line so that a false color around an edge in processed image can be reduced effectively. This mode is called "normal mode". These modes are selectable by a register setting. Actually, when the center of data is set to X_0, the following calculation is executed depending upon the interpolation mode.

$$x_0 = \begin{cases} (-X_{-2} + 2X_{-1} + 2X_0 + 2X_1 - X_2)/4 & \text{(normal mode)} \\ (X_{-1} + X_1)/2 & \text{(simple mode)} \end{cases}$$

Figure 10c shows an example of this horizontal interpolation processing in RGB Bayer CCD mode. In this figure, interpolated data is represented by small letters.

Figure 10d is a block diagram of horizontal interpolation module 1004. Two adders, one subtracter and a filter mode switch are implemented for executing one of these two types of filters. The filter mode switch is controlled by setting a register.

Vertical interpolation sub-module 1008 processes either a two-tap or three-tap vertical interpolation filter using two line-memories outside the preview engine module and outputs the information of all colors existing in the CFA pattern. And this sub-module also executes a vertical noise filter. An image processing in this module is a little complicated and the outputs from this sub-module is varied depending on a processed line, CCD mode, CFA pattern, filter mode and noise filter on/off. As explained in the following, the image processing from vertical interpolation sub-module 1008 to color adjustment sub-module 1014 has a strong correlation and this processing flow of them should be considered as a sequence of vertical interpolation processing. Therefore, this sequence of

the vertical interpolation processing is explained first. The sequence may be called "vertical interpolation sequence".

As with horizontal interpolation, vertical interpolation processing also has two types of interpolation mode, that is "simple mode" and "normal mode". An interpolation filter in simple mode utilizes the average two data at the next pixels on the upper and lower to interpolate the center of data. In normal mode, the processing differs between RGB CCD mode and complementary CCD mode. The interpolation filter in normal mode in RGB CCD mode utilizes the data of one of the others color same as horizontal interpolation filter. Actually, when the data of a certain color to be interpolated is set to X (mainly R,B) and the data of a color utilized as a reference is set to Y (mainly G), the following calculation is executed depending on the interpolation mode through this vertical interpolation sequence and it is the output from color adjustment sub-module.

$$x_0 = \begin{cases} (X_{-1} - Y_{-1} + X_1 - Y_1)/2 + Y_0 & \text{(normal mode)} \\ (X_{-1} + X_1)/2 & \text{(simple mode)} \end{cases}$$

Figure 10e shows an example of this vertical interpolation sequence for the RGB Bayer CCD pattern.

In complementary CCD mode, normal mode means "simple interpolation with color adjustment". That is, data of all colors which is processed by simple vertical interpolation is adjusted based on the formula in complementary color space. Actually, when the data of a certain color to be interpolated is set to X and the data of the others color is set to W, Y, and Z, the following calculations are executed in normal mode in complementary CCD mode.

$$x_0 = \begin{cases} (X_{-1} + X_1)/2 \pm a(w_0, x_0, y_0, z_0) & \text{(normal mode)} \\ (X_{-1} + X_1)/2 & \text{(simple mode)} \end{cases}$$

As to the calculation of $a = a(w_0, x_0, y_0, z_0)$, see below.

In this vertical interpolation sequence, main roles of vertical interpolation sub-module 1008 are to execute a part of vertical interpolation sequence and vertical noise filter. The part of vertical interpolation sequence means preparing data for normal vertical interpolation mode. As shown in Figures 10e and 10f (for RGB and complementary CCD patterns, respectively), in simple mode, an output data of this vertical interpolation sub-module bypasses color adjustment sub-module. Therefore, in simple mode, the output from this sub-module is used as the output of vertical interpolation sequence. In any case of interpolation mode, this sub-module calculates the following equation for vertical interpolation sequence.

$$x_0 = (X_{-1} - X_1)/2$$

Vertical noise filter ... which executes the following 3 taps vertical low pass filter is also processed in this sub-module depending on the CFA pattern.

$$x_0 = (X_{-1} - 2X_0 - X_1)4$$

However, for this filtering, data of same color on processed 3 lines must be prepared. Therefore, a function of the vertical noise filter mainly executes only G in RGB Bayer CCD. Figure 10g shows an example of the output of this vertical interpolation sub-module for a RGB Bayer CCD. When the vertical noise filter can be applied and it is set on, original data (R in this figure) is also adjusted in order to keep a correlation to the others color (G in this figure).

Figure 10h is a block diagram of vertical interpolation sub-module 1008. Six adders and two subtracters are implemented for executing vertical interpolation and noise filtering. Especially, a calculation process of L_121 and R_121 is so complicated that switching operation for L_121 and R_121 is not shown to simplify this figure.

Color selection sub-module 1012 arranges the inputs from vertical interpolation sub-modules in order of color format, that is R, G and B in RGB CCD mode or Ye, Cy, Mg, G in complementary CCD mode. This arrangement is executed automatically by setting register of the CFA pattern. Figure 10i shows an example of this color selection processing in RGB Bayer CCD of Figure 10g.

The outputs named "g" in this figure is a temporal data of G and is used for recalculation of R or B in RGB CCD mode in color adjustment sub-module.

Figure 10j is a block diagram of color selection sub-module 1012. Four color extractors switch and select independently correct colors from four inputs from vertical interpolation sub-module 1008.

Color adjustment sub-module 1014 executes the rest of calculation for vertical interpolation sequence. In RGB CCD mode such as RGB Bayer CCD, R or B is recalculated using the temporal data of G. When data of R or B from color selection sub-module is set to X, the following calculation is executed in RGB CCD mode.

$$x = X - G_{temp} + G$$

In the example of Figure 10i, when noise filter is off,

$$X = (b_{02} - b_{22})/2$$

$$G_{temp} = (G_{02} + G_{22})/2$$

$$G = g_{12}$$

Therefore,

$$\begin{aligned} x &= B \\ &= (b_{02} - b_{22})/2 - (G_{02} + G_{22})/2 + g_{12} \\ &= ((b_{02} - G_{02}) + (b_{22} - G_{22}))/2 + g_{12} \end{aligned}$$

This is the output B of the color adjustment module and also the output of vertical interpolation sequence. That is, vertical interpolation sequence in RGB CCD mode utilizes the average of differences between data of color to be interpolated and reference data of the others color.

In complementary CCD mode, color adjustment is processed to data of all colors from color selection sub-module. First, value a is calculated at each pixel based on a formula in complementary color space Ye + Cy = G + Mg.

$$a = G + Mg - Ye - Cy$$

That is, the value a can be considered as the amount of an error value of four colors. Therefore, in complementary CCD mode, to data of all colors, Ye, Cy, Mg and G, the following adjustment is processed to satisfy the above formula.

$$ye = Ye + a/4$$

$$cy = Cy + a/4$$
$$g = G - a/4$$
$$mg = Mg - a/4$$

Figure 10k is a block diagram of color adjustment sub-module 1014. Six adders and three subtracters are implemented for executing the two types of calculations described above. A switcher named CCDMOD in this figure selects correct outputs depending on CCD mode and is controlled by setting a register.

Comp2RGB conversion sub-modules 1016 converts complementary color formatted data to RGB formatted data in complementary CCD mode. Especially for G, data from color adjustment and data calculated by conversion formula can be blended by 5 types of blending ratio. Actually, the following calculation is executed based on the conversion formula:

$$R = Ye - Cy + Mg$$
$$G = rG_{input} + (1-r)(Ye + Cy - Mg) \quad (r=0,1/4,2/4,3/4,1)$$
$$B = Mg - Ye + Cy$$

In RGB CCD mode, data from color adjustment sub-module bypass this sub-module.

Figure 10l is a block diagram of comp2RGB conversion sub-module1016. Three adders, three subtractors, and two multipliers are implemented for executing the calculations above. A gain adjuster for G named "green_ratio" in this figure is adjustable by setting a register. In RGB CCD mode, a CCDMOD switcher selects off (high in this figure) for bypassing this module.

RGB gain for complementary CCD module allows adjustment of white balance by RGB color format even for complementary CCD module. This module is also available in RGB CCD mode.

Figure 9a is a block diagram of complementary white balance module 1408. One multiplier and clip circuit is implemented for this operation. Each gain for RGB is set by a register.

Gamma correction modules 1410 execute gamma correction for each color data in RGB color format. For this operation, prepare in advance three types of data for approximating the gamma curve by four linear segments.

Those are area, offset and gain shown in Figure 9b. As shown in Figure 14, this module exists for each color so that the independent adjustment to RGB may be made.

Figure 9c is a block diagram of gamma correction module 1410. Area detector selects correct gain and offset for input data based on area data. The data regarding gain, offset, and area are set in three registers.

RGB2YCbCr conversion module 1412 converts RGB formatted data to YCbCr formatted data and adjusts offsets to Cb and Cr based on the following matrix calculation.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} COEF1 & COEF2 & COEF3 \\ COEF4 & COEF5 & COEF6 \\ COEF7 & COEF8 & COEF9 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ OFFSET\_Cb \\ OFFSET\_Cr \end{bmatrix}$$

Each coefficient in this matrix is set by a register so that variable setting for this conversion is available.

Figure 11 is a block diagram of this RGB2YCbCr conversion module 1412. Nine multipliers and five adders are implemented for the foregoing matrix calculation. After multiplying RGB data with coefficients, the six least significant bits of each data from the multipliers is cut in order to reduce size of circuits. As to Cb and Cr, additional circuit for offset adjustment is followed by YCbCr conversion circuit. Clip circuits for Cb and Cr includes conversion circuits from two's complement to offset binary.

Burst mode compression/decompression engine

The preferred embodiment DSC engine includes an improved Burst Capture function with real-time processing, without compromise in the image resolution as compared to the regular capture mode. The Burst Capture Mode is the use of dedicated compression and decompression engine 108 for an increased burst capture sequence length. A sequence of CCD raw image frames is first stored in SDRAM 160 by using Compression engine 108. Then, as an off-line process, the image pipeline of regular capture mode retrieves the CCD raw images from SDRAM 160, processes them sequentially, and finally stores them

back as JPEG files in the SDRAM. The Animated Playback Mode can display these JPEG files.

Burst mode compression/decompression engine 108 includes differential pulse code modulation (DPCM) and Huffman coding using the same tables as the entropy-coding of DC coefficients in baseline JPEG compression. Engine 108 uses the sample Huffman table in the JPEG standard for chrominance DC differential data. Engine 108 also provides the inverse transforms as illustrated in Figure 13. Fixed Huffman Table (JPEG Huffman table for Chrominance DC coefficients):

Category (SSSS)	$\hat{D}_i$	Code Length	Codeword
0	0	2	00
1	-1,1	2	01
2	-3,-2,2,3	2	10
3	-7,..,-4,4,..,7	3	110
4	-15,..,-8,8,..,15	4	1110
5	-31,..,-16,16,..,31	5	11110
6	-63,..,-32,32,..,63	6	111110
7	-127,..,-64,64,..,127	7	11111110
8	-255,..,-128,128,..,128	8	111111110
9	-511,..,-256,256,..,511	9	1111111110
10	-1023,..,-512,512,..,1023	10	11111111110
11	-2047,..,-1024,1024,..,2047	11	111111111110
12	-4095,..,-2048,2048,..,4095	12	1111111111110

The encoder has four look-up tables: Huffman code (13 x 2-byte entries), Huffman code length table (13 x 1-byte entries), low bit mask to generate variable-length bit stream (32 x 4-byte entries), nad log table (256 x 1-byte entries). The Huffman tables are not programmable for simplicity, although alternative embodiments could include programmable Huffman tables.

The Huffman decoder performs the inverse function of the Huffman encoder and has five look-up tables: max code comparison table (13 x 2-byte entries), Min code comparison table (13 x 2-byte entries), decoded Huffman

symbol pointer (13 x 1-byte entries), decoded Huffman symbol table (13 x 1-byte entries), and bit position mask (32 x 4-byte entries).

The lossy mode compression just discards the least significant bit (LSB) or the two least significant bits of each coefficient.

Playback synchronization

A problem involved in playback of audio-visual bitstreams is how to synchronize audio with video signal. The preferred embodiments play the audio bitstream seamlessly in the background in real-time with the audio encoded by using the simple coding standards like ITU-T G.711 and Microsoft 16-bit PCM. By using an interrupt service routine, about 0.1% of the DSP resources is enough to output audio in real time through (multichannel) buffered serial ports; see Figure 1b. Therefore, the preferred embodiment must realize the video decoding in synchronization to the audio playback.

For clarity, assume that both audio and video are captured in full speed (real-time with 8K sample/s for audio and 30 frame/s for video). Audio is played back as samples. However, video is displayed in the granularity of frames. Thus the synchronization problem is caused by the fact that the video decoding could be faster or slower than the real-time requirement. If the video decoding is too fast, a certain amount of delay slots has to be inserted to slow down the decoding. Contrarily, if the video decoding is too slow, some video frames must be skipped to catch up with the real-time audio playback.

The preferred embodiments handle both cases. Especially in the case of slow video decoding, the preferred embodiments can properly select and skip the frames in an optimal manner. Note that the preferred embodiment is described for video bitstreams without bi-directional coded frames (B-frames).

Figure 46a depicts the synchronization between audio and video. The first video frame is pre-decoded before beginning audio-video playback. Since the video is displayed in the granularity of frames, the synchronization points are located at the video frame boundaries, i.e. $\{t = 0, \Delta T, 2\Delta T, 3\Delta T \ldots\ldots\}$. Here ΔT is the duration of a frame, which is defined as:

$$\Delta T = 1/fp \qquad (1)$$

where *fp* is the frame-rate used for the video sequence.

Audio and video could lose synchronization when the video decoding speed is not fast enough. As illustrated in Figure 46a, when the decoding of video frame 2 has not finished in time (Td2 > ΔT), the audio-video playback loses synchronization after displaying video frame 1. Here {Tdm, m =0,1,2,... ...} denotes the decoding time used for decoding video frame m.

With insufficient video playback speed, the only way to maintain a reasonable synchronization between audio and video is to skip video frames properly. In Figure 46b, video frame 2 is skipped (and frame 1 repeated) so that synchronization can be reacquired at frame 3.

A preferred embodiment circular buffer scheme is illustrated in Figure 47. The video decoder is connected to one side of the circular buffer, the display is connected to the other side. The circular buffer has a size of N video frames. There are two registers associated with each frame buffer of the circular buffer: the first register contains TP_n, n=0, 1, 2, 3, ..., N-1 which indicates the presumptive presentation time of the video frame stored in buffer n, and the second register contains S_n, n=0, 1, 2, 3, ..., N-1 which signals whether the frame in buffer n is ready for display (1 for ready, 0 for not ready). Of course, the value of TP_n is a multiple of ΔT. Buffer switching for display also occurs at frame boundaries (i.e. at time t = mΔT, m=0, 1, 2,...). Because the preferred embodiments use a circular buffer containing N frames, all the indices (..., n-1, n, n+1, ...) should be regarded as modulo-N indices.

Suppose the time after decoding the current video frame is T. The decoded current frame is stored in buffer n-1 in Figure 47. Therefore, the buffer to be used for storing the next frame in Figure 47 is buffer n.

Determine the current position in the bitstream: the frame index m of the current decoded frame is defined as

$$m = TP_{n-1} / \Delta T \qquad (2)$$

Determe the decoding starting time of the next frame: since the frame in the buffer n is to be displayed during the time interval of $\{TP_n \le t < TP_{n+1}\}$, buffer n is not

available for decoding the next frame until $\tilde{T}P_{n+1}$. Therefore, the decoding starting time of the next frame Ts is:

$$Ts = \max\{T, TP_{n+1}\} \qquad (3)$$

Determine the next frame to be decoded: let $\hat{T}d$ be the estimated time for decoding the next frame, the presentation time of the next frame must satisfy:

$$\begin{cases} TP_n > Ts + \hat{T}d \\ TP_n \geq TP_{n-1} + \Delta T \end{cases}$$

The above conditions imply that the decoding of the next frame is finished before its presentation time, and the next frame is located at least a frame after the current frame in the bitstream. Because TP_n must be a multiple of ΔT, the next frame that can be synchronized to audio satisfies the conditions:

$$\begin{cases} TP_n = \Delta T \left[\dfrac{Ts + \hat{T}d}{\Delta T} + 0.5 \right] \\ TP_n \geq TP_{n-1} + \Delta T \end{cases}$$

where [•] denotes integer part by truncation.

Therefore, the presentation time of the next frame is determined by:

$$TP_n = \max\left\{ \Delta T \left[\frac{Ts + \hat{T}d}{\Delta T} + 0.5 \right], TP_{n-1} + \Delta T \right\} \qquad (4)$$

There are different methods to estimate $\hat{T}d$, such as using statistical estimation based on prior decodings or frame parameters. One preferred embodiment simply uses the actual decoding time of the most recently decoded frame of the same picture coding type (I-frame or P-frame) plus a certain amount of safety margin as the estimated decoding time for the next frame.

The frame index m' of the next frame to be decoded can thus be computed as:

$$m' = TP_n / \Delta T \qquad (5)$$

Then the number of frames Δm to be skipped from the current position is determined by:

$$\Delta m = m' - m - 1 \qquad (6)$$

Equation (2) to (6) make up of the basic control operations for updating the circular buffer.

The preferred embodiments use the circular buffer scheme to realize synchronization. There are two parts: the video decoder buffer switch control and the display buffer switch control. Figure 48 demonstrates the flowchart of the video decoder buffer switch control, which contains two stages: initialization and playback.

Initialization: in the circular buffer initialization, N_f ($1 \le N_f \le N$) video frames are decoded before starting playback. As shown in the dashed box in Figure 48, there are four steps for the initialization:

- step 0: set all the presentation time registers {TP_n, n=0, 1, 2, 3, ..., N-1} and the status registers {S_n ,n=0, 1, 2, 3, ..., N-1} to zero, switch the video decoder to buffer 0 (i.e. n=0), point to the beginning of the video bitstream (i.e. m' = Δm = 0), set time to zero (i.e. t =0)
- step 1: set the related status register S_n to 1, skip Δm video frames, decode frame m', store the decoded frame in buffer n. (Recall on the first pass through the loop, n=0, m'=0, so the first frame is decoded and stored in buffer 0.)
- step 2: set the decoding start time Ts to t, switch to the next buffer (i.e. n++), update TP_n, m', Δm according to equations (4), (5), and (6).
- step 3: check whether the number of decoded frames reaches the pre-set frame number N_f. If true, go to playback, otherwise, loop to step 1.

Playback: there are six steps involved in updating the circular buffer during the playback.

- step 0: switch display to buffer 0, enable display, reset time to zero (i.e. t = T =0), switch the video decoder to buffer N_f (i.e. n = N_f)
- step 1: if the whole video sequence is decoded, stop decoding, otherwise, go to step 2.

- step 2: update Ts, TP_n, m' and Δm according to equations (3), (4), (5), and (6).
- step 3: wait until time reaches Ts (i.e. t ≥ Ts), go to step 4.
- step 4: set the related status register S_n to 0, skip Δm video frames, decode frame m', store the decoded frame in buffer n.
- step 5: if the frame decoding finishes in time (i.e. t < TP_n), set S_n to 1 to indicate the decoded frame is ready for display, set T to t, switch the video decoder to the next buffer (i.e. n++). Otherwise, set T to t, add DT to the estimated $\hat{T}d$ (i.e. $\hat{T}d += DT \quad with \quad DT = N_d \Delta T$, intentionally skip N_d ($0 \le N_d$) more frames in the next stage), set the current frame index m to m'. Go to step 1. Note that N_d is a parameter to control the screen freezing time before resuming the synchronization.

Users can freely decide the circular buffer size (N) , the initial time delay (N_f) for the playback as well as the screen freezing time (N_d). Obviously, the minimum buffer size is 3 video frames (i.e. N = 3), the least time delay is one video frame (i.e. N_f = 1). However, in the case of insufficient video decoding speed, it is strongly recommended to decode N-1 frames (i.e. N_f = N-1) during the circular buffer initialization, so that the video decoder can gain the maximal room to catch up with the audio real time playback.

Display buffer switch control: the display buffer switch control is carried out in parallel to the video decoder buffer switch. The preferred embodiment checks the display buffer switch at video frame boundaries: t = mΔT, m=0, 1, 2, ... Suppose the display is currently showing the video frame in buffer n-1, it switches to the next buffer, i.e. buffer n, if and only if the current time ($t \ge TP_n$) and (S_n=1) holds. Otherwise, it is connected to buffer n-1. Here, if ($t \ge TP_n$) and (S_n=0), it means the decoder has not finished decoding of the frame in time. In this case, the video frame in buffer n has been discarded, the decoder is decoding the conservatively selected next frame to update buffer n again, the display should keep displaying the frame in buffer n-1 until ($t \ge TP_n$) and (S_n=1) holds.

In summary, the preferred embodiment provides a way to realize the synchronization between audio and video when playing back by using software or firmware.

Variable length decoding

Variable Length Decoding (VLD) is involved in decoding bitstreams which are generated by using Variable Length Encoding (VLC) at encoder; see Figure 1b item 126. Because of VLC, the number of bits used for coding units varies from unit to unit. Therefore, a decoder does not know the number of bits used for a coding unit before having decoded it. This makes it essential for a decoder to use a bitstream buffer during the decoding process.

In video coding, for example, a frame to be encoded is decomposed into a set of macroblocks (see Figure 49). Under the consideration of the smallest memory requirement, a coding unit here is normally defined as macroblock, which consists of a 16x16 pixel luminance area and the corresponding chrominance areas depending on the chroma format (4:2:0, 4:2:2:, or 4:4:4). Certainly, a slice (a row of macroblocks in a frame) or even the frame itself can be treated a coding unit if there is enough memory.

Figure 50 depicts the video playback on a preferred embodiment digital still camera (DSC). In DSC applications, the video bitstream is pre-captured and stored on the high-capacity SDRAM, and the video decoder is built on the DSP. Since it is extremely expensive for the decoder to directly access the SDRAM, an on-chip bitstream buffer is opened on the DSP internal memory. The bitstream is first loaded from SDRAM to the bitstream buffer through the SDRAM, then the decoder uses the bitstream in the bitstream buffer to reconstruct video. Since the bitstream loading is achieved by using DMA (Direct Memory Access), which can run in the background without intervention of a CPU, the bitstream loading overhead is mainly due to time used for setting up registers for the DMA transfer.

There are two basic requirements in terms of bitstream buffer management. First of all, the buffer size should be big enough to cover the worst case. For example, in video coding, the theoretically maximal number of bits for encoding a macroblock could be 256 words (one word here is defined as two bytes). Although this worst case is very rare, the bitstream buffer size has to be 256 words in order to be at the safe side. Secondly, the bitstream buffer should never underflow, that is,

the buffer management should guarantee that the bitstream for a coding unit is available when it is being decoded.

There are different schemes to satisfy the second requirement. The simplest one would be to check the decoding position in the bitstream buffer at each buffer access. The bitstream buffer is re-filled whenever the decoding position is out of the valid buffer range. Because the decoding is a bit by bit operation, this scheme is not realistic: it spends too much overhead in deciding when to re-fill the buffer.

A realistic scheme is the linear shifting buffer scheme as shown in Figure 51a. In this scheme, the bitstream buffer is linearly accessed by the decoder from left to right, after decoding a unit the rest of the bitstream is shifted forward to the beginning of the buffer, then the buffer is re-filled to "full" before decoding the next unit. In Figure 51a, Ps and Pd denote the current decoding position and the bitstream end position in the bitstream buffer, respectively.

This buffer scheme has two disadvantages. First, since the buffer size is much larger than the average number of bits of the decoding units, a lot of time will be spent on the bitstream shifting. For instance, in video decoding the buffer size is 256 words to cover the worst case, but on average a unit may only use 16 words, this means about 240 words of shifting for each unit. The second disadvantage is that it requires a bitstream loading after decoding each unit; this costs additional overhead because time has to spent on issuing the DMA transfers.

A better buffer management scheme is so-called quasi-circular buffer scheme as shown in Figure 51b. In this scheme, the decoder accesses the bitstream buffer in a circular manner. This avoids the bitstream shifting required by the linear buffer scheme. There are two cases after decoding a unit. This first case is in the lefthand portion of Figure 51b: the rest of bitstream is located in the middle of the buffer. In this case, the buffer is filled by loading the bitstream twice, one for the right end followed by the other one for loading the left end. (Note: if the bitstream loading can write the bitstream into the bitstream buffer in a circular manner, only one load is needed; however, this is not always the case.) The

second case is shown in the righthand portion of Figure 51b, in which only the middle of the buffer needs to be filled.

The quasi-circular buffer scheme is much more efficient than the linear shifting buffer because it avoids bitstream shifting, but it still suffers from a disadvantage that one or two bitstream loads are needed after decoding each unit. The following preferred embodiment hybrid circular-double buffer scheme solves this problem.

Figure 52 status 0 shows a hybrid circular-double buffer containing two buffers of equal size; namely, the left buffer and the right buffer. There is a flag for each buffer to indicate the buffer fullness ("full" / "not-full"). Ps points to the current decoding position after decoding a unit. In terms of buffer size, each buffer covers the worst case of decoding coding units, this makes the hybrid buffer size twice of a linear shifting buffer or a quasi-circular buffer. Unlike a traditional double buffer, the two buffers here have a continual memory allocation, i.e. the left buffer is directly followed by the right buffer in the memory map. The decoder accesses the hybrid buffer in a circular manner.

The preferred embodiment hybrid buffer operates through the following four statuses:

- Status 0: the initialization status, both the left and right buffers are fully loaded and set to "full", Ps points to the beginning of the hybrid buffer.
- Status 1: after decoding the first unit, change the left buffer flag to "not-full".
- Status 2: after decoding a unit, if the current decoding position Ps is in the right buffer and the left buffer flag is "not-full", fully load the left buffer and set the left buffer flag to "full". In addition, if the right buffer flag is "full", change it to "not-full". Otherwise, no action is taken.
- Status 3: after decoding a unit, if the current decoding position Ps is in the left buffer and the right buffer flag is "not-full", fully load the right buffer and set the right buffer flag to "full". If the left buffer flag is "full", change it to "not-full". Otherwise, no action is taken.

Taking the preferred embodiment platform (e.g., Figure 1b) as an example (where data is in 16-bit units), define the following data type:

```
typedef struct bitstream {
      SInt bit_ptr;      /* current bit position (0 ~ 16)
       */
      SInt Ps;           /* current decoding position in bitstream
buffer */
      SInt left_flag     /* left buffer flag "full / not-full"
       */
      SInt right_flag    /* right buffer flag "full / not-full"
       */
      USInt *databuf;    /* bitstream buffer
       */
      Long Addr_SDRAM;   /* bitstream address in SDRAM
*/
} Bitstream;
```

The pseudo code shown in Table 1. describes the hybrid circular-double buffer scheme. Function BufferInitialization() is called only once at the beginning of decoding, while function BitstreamBufferUpdate() is called after decoding each coding unit, it automatically updates the buffer flags and re-loads the buffers if the conditions become true. In Table 1 BUFSIZE stands for the buffer size of the hybrid circular-double buffer.

```
Void BufferInitialization(
        Bitstream *stream,        /* pointer of bitstream                    */
        )
{

        /*=============================================================*/
        /* Initialization of the hybrid circular-double buffer         */
        /*=============================================================*/
        LoadBuffer(&stream->databuf[0], stream->Addr_SDRAM, BUFSIZE);
        stream->Addr_SDARM +=BUFSIZE;
        stream->left_flag = "full";
        stream->right_flag = "full"
        stream->Ps  = 0;
        stream->bit_ptr = 16;
}


Void BitstreamBufferUpdate(
        Bitstream *stream,        /* pointer of bitstream                    */
        )
{

        /*=============================================================*/
        /* Update the left buffer if necessary                         */
```

```
        /*=================================================================*/
        if (stream->left_flag =="not-full"  && stream->Ps >= BUFSIZE/2)
        {
          LoadBuffer(&stream->databuf[0], stream->Addr_SDRAM. BUFSIZE/2);
          stream->Addr_SDARM +=BUFSIZE/2;
          stream->left_flag = "full";
        }


        /*=================================================================*/
        /* Update the right buffer if necessary                            */
        /*=================================================================*/
        if (stream->right_flag =="not-full"  && stream->Ps < BUFSIZE/2)
        {
          LoadBuffer(&stream->databuf[BUFSIZE/2], stream->Addr_SDRAM, BUFSIZE/2);
          stream->Addr_SDARM +=BUFSIZE/2;
          stream->right_flag = "full";
        }

        /*=================================================================*/
        /* Update the left buffer flag                                     */
        /*=================================================================*/
        if (stream->left_flag =="full"  && stream->Ps < BUFSIZE/2)
          stream->left_flag = "not-full";

        /*=================================================================*/
        /* Update the right  buffer flag                                   */
        /*=================================================================*/
        if (stream->right_flag =="full"  && stream->Ps >= BUFSIZE/2)
          stream->right_flag = "not-full";

}
```

Table 1. Pseudo code for the hybrid circular-double buffer scheme

As it can be seen in BitstreamBufferUpdate() in Table 1, the left buffer or right buffer is not reloaded after decoding each unit, but is loaded only if the opposite buffer (left / right) is in use and its buffer flag is "not-full". This greatly reduces the number of buffer loads. Consider the video coding as an example. This needs BUFSIZE of 512 words if a macroblock is the unit, the average bitstream size of a unit is assumed to be 16 words. Because the linear shifting buffer and the quasi-circular buffer re-fill the buffer after decoding each unit, the average loading length for those two schemes is also 16 words. Compared with the fixed loading length of 256 words in the hybrid circular-double buffer scheme, the preferred embodiment reduces the loading overhead by a factor about 16 (i.e. 256 / 16).

Mini-experiments compared the three buffer schemes discussed above. The video sequence used was coastguard (352 x 288, 300 frames, 4:2:0). The bitstream is generated by using a MPEG1 video encoder. The target bit-rate is 3 Mbit/s, I-frame only. The same decoder with three different buffer schemes are used to decode the same bitstream, the buffer loading count and word shifting count are recorded during the decoding. The performance comparison among the three buffer schemes is listed in Table 2. As shown in Table 2, for each macroblock the linear shifting buffer scheme requires one buffer load, and on average about 240 words of shifting. The quasi-circular buffer scheme needs slightly more buffer loads (1.06 load/macroblock) but no shifting. The preferred embodiment hybrid circular-double buffer scheme used only about 0.0619 buffer load per macroblock. On the preferred embodiment platform of Figure 1b in particular, the preferred embodiment scheme provides a cycle count reduction ratio of about 113 and 17 in comparison to the linear shifting buffer scheme and the quasi-circular buffer scheme, respectively.

	Linear shifting buffer	Quasi-circular buffer	Hybrid circular-double buffer
Buffer size (words)	256	256	512
Number of loads per macroblock	1.00	1.06	0.0619
Number of word shifting per macroblock	240.15	0	0
Overhead per load (cycles)	80	80	80
Cycle count per word shifting	2	2	2
Total cycles used for bitstream buffer per macroblock	560.30	84.72	4.95
Cycle count ratio vs. the hybrid circular-double buffer scheme	*113.19*	*17.12*	*1.00*

Table 2. Performance comparison among three buffer schemes on TMS320DSC21 platform

Onscreen display and graphics acceleration

The Onscreen display (OSD) module 105 is responsible for managing OSD data from different OSD windows and blending it with the video. It reads OSD data from SDRAM 160, and outputs to NTSC/PAL encoder 106. The OSD module defaults to standby mode, in which it simply sends video to NTSC/PAL

encoder 106. After being configured and activated by ARM CPU 130, the OSD module reads OSD data and mixes it with the video output. ARM CPU 130 is responsible for turning on and off OSD operations and writing the OSD data to the SDRAM. Figure 15 shows the block diagram of the OSD module and affiliated other items. The various functions of the OSD are described in the following paragraphs.

OSD data storage. The OSD data has variable size. In the bitmap window, each pixel can be 1, 2, 4, or 8 bits wide. In the YCrCb 4:2:2 window, it takes 8-bit per components, and the components are arranged according to 4:2:2 (Cb/Y/Cr/Y ...) format. In the case where RGB graphics data needs to be used as OSD, the application should perform software conversion to Y/Cr/Cb before storing it. The OSD data is always packed into 32-bit words and left justified. Starting from the upper left corner of the OSD window, all data will be packed into adjacent 32-bit words.

Setting up an OSD window. An OSD window is defined by its attributes. Besides storing OSD data for a window into SDRAM by ARM CPU 130, the application program also needs to update window attributes and other setup in the OSD module as described in the following subsections.

Location register. The Location register contains X and Y locations of the upper left and lower right corners of each window. The application program needs to set up the CAM and enable selected OSD windows; see Figure 16.

Color look up tables. The OSD has the fixed 256-entry color look up table (CLUT). The CLUT is used to convert bitmap data into Y/Cr/Cb components. In the case of 1,2 or 4 bitmap pixels, the CLUT can be determined by CLUT registers.

Blending and transparency. Color blending on the pixel level is also supported. This feature is available for the bitmap displays only (Window1,2). If the window color blending is enabled, the amount of blending of each pixel is determined by the blending factor. As shown in the following table, the window blending supports 5 different levels, according to the selected blending factor. The hardware also supports a transparency mode with bitmap. If transparency is

enabled, then any pixel on the bitmap display that has a value of 0 will allow video to be displayed. Essentially, 0-valued pixels are considered the transparent color, i.e. the background color will show through the bitmap. The Table shows the connection between transparency and blending on the same window.

Transparency	Blend Factor	OSD window contribution	Video contribution
OFF	0	0	1
	1	1/4	¾
	2	1/2	1/2
	3	3/4	1/4
	4	1	0
ON		if pixel value = 0	if pixel value = 0
	0	0	1
	1	1/4	3/4
	2	1/2	1/2
	3	3/4	1/4
	4	1	0

Hardware cursor. A rectangular shape is provided using hardware window1. With window1, the cursor always appears on top of other OSD Windows. The user can specify the size, color of the shape. When hardware window1 is designated as the cursor, only two windows are available for the OSD application. If a hardware cursor is not used, then the application can use window1 as a regular hardware window. Figure 17 shows an example of the hardware cursor.

DSP subsystem

The DSP subsystem consists of C54x DSP, local memory blocks, iMX and VLC accelerators, shared image buffers, and the multiplexers implementing the sharing.

C54x is a high performance, low power, and market proven DSP. cDSP hardware and software development tools for C54x are also very mature.

The DSP carries out auto exposure, auto focus, auto white-balancing (AE/AF/AWB) and part of the image pipeline tasks. It also handles SDRAM transfer and drives the accelerators to implement the rest of image processing and image compression tasks. Flexibility and ease of programming in the DSP enables

camera makers to refine the image processing flow, adjust quality-performance tradeoffs, and introduce additional features to the camera.

The configurable DSP (cDSP) design flow is adopted to allow flexibility and design reuse. The memory blocks time-shared among DSP and accelerators are large enough for one processing unit (16x16 pixels) and provide zero-wait state access to DSP.

Features

Fixed-point Digital Signal Processor
100 MIPs LEAD2.0 CPU
On-module RAM 32Kx16bit
(4 blocks of 8Kx16bit dual access program/data RAM)
Multi-Channel Buffered Serial Ports (McBSPs)
ARM can access RAM via Enhanced 8-bit Host Port Interface
One hardware timer
On-chip Programmable PLL
Software Programmable Wait-State Generator
Scan-based emulation and JTAG boundary scan logic

Figure 18a shows more details on the DSP subsystem and in particular the details of the connection between the DSP and the iMX and VLC. Figure 18b is the memory map.

The shared memory blocks A and B occupy two 2Kword banks on the DSP's data memory space. Each block can be accessed by DSP, iMX, VLC, and SDRAM controller depending on static switching controlled by DSP. No dynamic, cycle-by-cycle, memory arbitration is planned. DSP's program should get seamless access of these memory blocks through zero-wait-state external memory interface.

The configuration memory blocks, for iMX coefficient, iMX command, VLC Q-matrix, and VLC Huffman table, also connect to DSP's external memory interface. They are also statically switched between the specific module and DSP. Typically at power-up or at initial stage of camera operation mode, these memory blocks are switched to DSP side so DSP can set up the appropriate configuration information for

the operation. Then, they are switched over to iMX and VLC for the duration of operation.

Imaging Extension (iMX)

iMX, imaging extension, is a parallel MAC engine with flexible control and memory interface for extending image processing performance of programmable DSPs. iMX is conceived to work well in a shared memory configuration with a DSP processor, such that flexibility, memory utilization, and ease of programming are achieved. The architecture covers generic 1-D and 2-D FIR filtering, array scaling/addition, matrix multiplications (for color space transform), clipping, and thresholding operations.

For digital still cameras, iMX can be used to speed up
CFA interpolation,
color space conversion,
chroma down-sampling,
edge enhancement,
color suppression,
DCT and IDCT,
Table lookup.

iMX methodology originates from the discipline of parallel processing and high performance computer architecture. The design comprehends the need for a scalable MAC engine. iMX in the first preferred embodiment incorporates 4 MAC units; see Figure 19. Alternative preferred embodiments upgrade to 8 MAC units or more. Software can be structured so that the hardware upgrade will not incur substantial software changes.

Much flexibility of iMX is due to parameter-driven address generation and looping control. Overall efficiency comes from efficient pipelining control inside iMX as well as the system-level memory buffering scheme.

iMX works best for block-based processing. To facilitate this, the datapath needs to connect to data input/output and coefficient memory. iMX contains data

input, data output, and coefficient memory ports, and allows arbitration among these ports. This eliminates the need for dedicated memory blocks, and brings more flexibility and better memory utilization on the system level. These memory blocks are accessible as DSP data memory to facilitate data exchange.

There is a separate command memory that feeds a command decode unit in iMX. The command memory should be specified to fit all the accelerated steps in our reference image pipeline algorithm, so that this sequence of commands can be executed with little intervention from DSP.

iMX block diagram appears in Figure 20. A command decode subblock reads and decodes commands, and drives static parameters, one set per command, to the address generator. Address generator then computes looping variables and data/coefficient/output pointers, and coordinates with execution control, which handles cycle-by-cycle pipelining control. Address generator sends data and coefficient read requests to the arbiter. Arbiter forwards the requests to the data/coefficient memory. Data read back from memory go to the input formatter, which takes care of data alignment and replication. Formatted data and coefficients are then provided to the datapath, which mainly consists of the 4 MAC units. Output from datapath is routed to arbiter for memory write.

iMX communicates to DSP via shared memory (for data input, coefficient, data output, command) and via memory-mapped registers (start command, completion status). All data buffers and memory blocks are single-ported, and are switched to one party or another via static control, rather than on-line arbitration.

In a typical application, DSP would place filter coefficients, DCT/IDCT cosine constants, and lookup tables in the coefficient memory, and put iMX commands in the command memory. DSP then turns over access to these memory blocks to iMX. These memory blocks are sized adequately for our reference design to fit all needed coefficients and commands for a major camera operation mode (e.g., image capture). Any update/reload should occur very infrequently. In case either or both memory blocks run out of space, paging can be performed.

DSP manages the switch network so that, to iMX, there is only one data buffer. During run time, DSP switched the A/B buffers among itself, iMX. VLC, and SDRAM controller to implement data passing.

Figure 21 illustrates a simple table lookup accelerator with input rounding/clipping capability used to speed up the image pipeline on the DSP. This is carried out with a very simple control structure and datapath.

VLC engine

VLC accelerator is a coprocessor optimized for quantization and Huffman encode in the context of JPEG compression and MPEG compression. It operates with quantizer matrices and Huffman tables preloaded by DSP, via shared memory blocks. Aggressive pipelining in the design achieves very high throughput rate, above 30 million DCT coefficients for compression.

VLC's working memory, including quantizer matrices, Huffman tables, and data input/output memory, are all shared memory blocks.

VLC functionality

Basically, VLC covers Quantization, zigzag scan, and Huffman encode for JPEG encode (baseline DCT, 8-bit sample), with up to 4 quantizer matrices (stored as $invq[i,j] = 2^{16}/q[i,j]$) and 2 encode Huffman tables all loadable. Can process one MCU that contains up to 10 blocks. Each block consists of 8x8 = 64 samples.

Quantization, zigzag scan, and Huffman encode for MPEG-1 video encode. One macroblock, with up to six 8x8 blocks, can be processed. Number of blocks and within them, number of luminance blocks, can be specified. Huffman encode can be bypassed to produce quantized and zigzag-ordered levels.

The accelerator requires memory blocks for input/output buffer, quantization matrices and Huffman encode tables. The memory configuration should be sufficient to support normal encode operations, one JPEG MCU (minimum coding unit), or MPEG macroblock per call.

Both input and output must fit the 2K words (1word = 16-bit) shared memory buffer (A or B). MCU or macroblock has maximally ten 8x8 blocks, or 640 input words. Compressed output data is typically smaller than input size.

JPEG Huffman encode table takes up (12 x 176) x 32-bit, or 384 words per table. JPEG standard allows 2 tables, so taking totally 768 memory words. MPEG tables are hard-wired into VLC and do not take up memory. We have allocated 2K words for the Huffman tables.

The quantizer matrix memory, 512 words by 16-bit, allow for 8 quantizer matrices to coexist, each taking 64 x 16-bit. JPEG allows for 4 matrices, and MPEG encode requires 2 matrices.

Figure 22 shows the major subblocks of VLC. Only the encode path is implemented in one preferred embodiment VLC module; alternative preferred embodiments incorporate the decode path into the module.

ARM subsystem

ARM microprocessor 130 handles system-level initialization, configuration, user interface, user command execution, connectivity functions, and overall system control. ARM 130 has a larger memory space, better context switching capability, and is thus more suitable for complex, multi-tasking, and general processing than DSP 122. Preferred embodiments integrate an ARM7 cTDMI core; see Figure 1b. ARM7 core is specified up to at least 40 MHz. The ARM subsystem will also have a 32 Kbytes local static RAM 132.

ARM processor 130 is connected to all the DSC peripherals including CCD Controller, TV encoder, preview engine, IrDA, USB, Compact Flash/Smart Media, UART, etc.

ARM processor 130 is involved with the management of CCD incoming raw data and intermediate data to the SDRAM and LCD. Connected to all I/O devices, the ARM manages and is responsible for the smart devices such as USB, IrDA, Compact Flash/Smart Media, and UARTS. The four basic operation modes of PREVIEW, CAPTURE, PLAYBACK, and BURST are initiated by requests from the ARM. The ARM will then monitor the device for completion of the request and in some cases will manage data after the request is completed.

After RESET and before any of the camera operations can occur, the ARM must perform several housekeeping tasks. The intial task is known as the BOOT

operation task. This function not only initializes the I/O and peripherals to a known state, it also must prepare, load and start DSP 122. This sequence begins by reading the DSP boot code from the flash, loading the DSP code memory and then releasing the DSP from its HOLD state. Additional DSP code is loaded into the SDRAM in a format the DSP can then read and overlay into its code space without ARM intervention.

ARM SDRAM Interface

ARM has two types of access to the SDRAM (1) through SDRAM buffer (burst read/write) and (2) direct access to the SDRAM with a higher latency - 4 cycle READ, 6 cycle WRITE. The direct access to memory can be word, half word or byte access.

The ARM/SDRAM controller interface also has a 32 byte buffer. The SDRAM burst request first fills this buffer and ARM reads and writes from/to this buffer.

ARM External Memory Interface

ARM 130 connects to the external memory through the External memory interface module. ARM 130 connects to the Compact Flash/Smart media through this interface. ARM 130 also connects to the off chip flash memory through this interface. DMA block (Figure 1b) enhances the ARM to CF/Smart media transfer.

ARM/DSP BOOT Sequence

The DSP BOOT sequence begins after a power up or after a COLD START. In this state, DSP 122 is in a HOLD condition waiting on initialization from ARM 130. The ARM checks DSP status registers to assure the DSP is in a HOLD state. The ARM programs the DSP boot code data to the DSP code memory from the FLASH.. The code is organized in logical overlays that allow the ARM to select the proper code for the function needed, in this case BOOT code.

The ARM loads the DSP code using the HPI Bridge (HPIB) interface. This interface can be programmed to access in either 8- or 16-bit width. For BOOT purposes, this will always be a 16-bit access.

After the code is loaded, the ARM signals the DSP to begin by releasing the HOLD. The DSP then begins its reset sequence from an address of DSP 7F80h

which is in the DSP RESET vector area. Upon completion of the RESET sequence, the DSP then branches to DSP FF80h, which is the beginning of the BOOT program loaded by the ARM.

Figure 23a shows the data paths used in the ARM/DSP boot sequence as well as data, request and command exchanges discussed later.

Capture Mode

ARM 130 programs CCD controller 102 to capture an image. The CCD controller auto transfers the image data to SDRAM and interrupts the ARM using IRQ1 when the transfer is complete. The ARM then notifies the DSP the RAW picture data is available to crunch. When the processing of the raw data is complete, the DSP signals the ARM the task is finished.

Preview Mode

The CCD will be programmed for a 30 fps high frame rate but reduced resolution vertically. The reconfiguration of the CCD and TG (timing generator) will cause the raw picture data to go to preview engine 104. The DSP will post process the data in SDRAM and prepare parameters for FOCUS, EXPOSURE and WHITE BALANCE. The ARM is signaled by the DSP when new adjustment parameters are ready and those corrections are applied by the ARM. The transferring of the correction parameters use the same communication interrupt architecture as previously mentioned and are expected to be at the current frame rate.

Burst Mode

The burst mode timing is based on the ARM clocking the picture rate from application parameters. Similar to a cross between Capture and Preview modes, the ARM programs the CCD for a capture that stores a compressed image into SDRAM through the compression engine. As in Preview mode, the ARM receives adjustment parameters from the DSP to make corrections of FOCUS, EXPOSURE and WHITE BALANCE.

Idle Mode

ARM may use an idle mode to receive correction parameters from the DSP during periods preceding other camera modes. If not in a power down situation, this time of 10-15 frames will allow the DSP-to-ARM correction loop to make auto

corrections on FOCUS, EXPOSURE and WHITE BALANCE. This idle mode will simulate Preview mode for the purposes of obtaining a stable correction.

ARM/DSP communication

The communication between ARM 130 and DSP 122 is via the HPIB (Host Port Interface Bridge). The HPIB physically connects the DSP (a C5409 type DSP) ports and BUSC (BUS Controller) 134. The ARM accesses the DSP memory by programming the HPIB, opening a 32k-word window into the DSP memory map. The map contains the data structures shared by the ARM and DSP for command request's, acknowledgements and datagrams.

The HPIB contains five sub-blocks. They are the interface, timing generator, DSP control registers, and interrupt hold sections.

The interface section receives and stores data from BUSC 134 and transfers it to and from the C5409. This interface can be an 8- or 16-bit data path to the C5409 and is 16-bit to the BUSC. An added feature is the ability to exchange the upper and lower byte if programmed to do so.

The timing generator makes signals HBIL and HDS and detects signal HRDY. HBIL is the HPI byte identification signal to the C5409. The HDS is the data strobe signal to the C5409 and the HRDY is the ready signal read from the C5409.

The interrupt hold section will detect the HINT level and make the INTC pulse synchronized with the ARM clock. The module will also set the HOLD port of the C5409 and detect HOLDA.

In 8-bit mode, address data from the ARM will not reach the C5409. The address is used only if the C5409 internal memory is selected. Therefore, the ARM must set the address in the HPIA register before sending or receiving data to the 32 Kword DARAM. The 8-bit mode may also be used for ARM<->DSP handshaking. The ARM will use the HINT bit in the HPIC register to interrupt the C5409.

In 16-bit mode, the HPIA/HPIC/HPID are not used. The ARM can access the C5409 internal memory as if it exists in the HPIB module. This mode will deliver faster performance, but does not support the HANDSHAKE signals because of these are routed in the HPIC register.

Figure 23b shows the signals and paths for the ARM to reach the C5409 DARAM.

Figure 23c indicates the shared memory map between the ARM (HOST) and the C5409 processor. When the ARM selects the memory area, "DSP Memory", BUSC takes cs_hpib signal active. The ARM can now access the DSP internal memory (32 kword DARAM + HPIA + HPIC + HPID).

When the ARM selects the "DSP Controller" area, BUSC takes cs_dspc signal active. The ARM is now accessing registers related to the C5409.

Multi-processing debugging environment

The preferred embodiment integrates ARM 130 and DSP 122 and thus multi-processing and thus requires debugging and development support. The preferred embodiment accomplishes this with a single JTAG connector 170 with additional emulation logic as illustrated in Figure 24.

Input/Output modules

The input/output module provides the different interfaces with the DSC peripherals as follows.

TV encoder 106 produces NTSC/PAL and RGB outputs for the LCD display and TV.

CCD/CMOS controller 102 generates timing signals VD/HD, can synchronize on externally generated HD/VD signals (#0 of MODESET register, #0 of SYNCEN register), supports progressive scan and interlaced CCDs, generates black clamping ontrol signals, programmable culing pattern 9CULH, CULV registers), 1 line/2 line alternating fields, MCLK (generated by CCD module), WEN (WRQ on TG, active-high) indicates CCD controller writing data to SDRAM, TG serial port interface (clk, data, TG chip select) is controlled by GIO pins, Iris, mechanical shutter, focus and zoom are controlled by GIO pins.

USB 142 from programmer's perspective consists of three main parts: FIFO controllers, UDC controller, and UDC core. USB configuration: INTERFACED0 ALT0 ENDPOINT0: CONTROL; INTERFACE0 ALT0 ENDPOINT1: BULKIN; INTERFACE0 ALT0 ENDPOINT1: BULKOUT; INTERFACE1 ALT0 ENDPOINT2: ISOIN; INTERFACE2 ALT0 ENDPOINT3: INTERRUPT IN. Buffer configuration:

SUB module has six FIFOs inside; each FIFO is of the same construction, except for direction and buffer size; USB module has only one unified memory for all endpoints; buffer sizes are programmable as long as all buffers fit inside the memory.

UART part of I/O block 140, supports start/stop communication protocol, detects parity errors (supporting dta length of 7 or 8 bits with even, odd, or no parity and 1 or 2 stop bits), has 32 bytes of FIFO for both transmitter and receiver, generates interrupts for a FIFO overflow or a time-out is detected on data receiving. ARM 130 control UART modules. There are seven 16-bit width registers which are accessible from ARM 130: data transmitter/receiver register (FIFO), bit rate register, mode register, FIFO control register for receiver, FIFO control register for transmitter, line control register, and status register. Figure 25 is a block diagram.

Compact Flash/Smart Media interface 180 is used to save/store image or user's data to a compact flash card or smart media; see Figure 26. The interface supports two kinds of operation modes for register setting and data transfer: memory mapped mode and I/O mode. An ARM 130 interrupt is generated for card detection while a compact flash card is being plugged or unplugged. The pins for both the smart media and the compact flash control interfaces are overlapped and can be switched by ARM 130 depending on product needs; see Figure 26.

In particular, the compact flash controller has registers mapped to the ARM memory space. The compact flash controller is responsible for generating the related control signals to the interface pins, and writes at 420 KB/s and reads at 2.0 MB/s. SDRAM can be utilized for storing at least one picture and an attempt to write to the compact flash with a big sector count, as done in a DOS machine, will invoke the fast write performance.

In contrast, the smart media controller has five register settings: command register, address1 register, address2 register, address3 register, and data port register. These five registers are mapped to the ARM memory space, and smart media controller wil generate the related signals for different register access automatically.

Audio input/output may be through the serial port of I/O block 140 with DSP buffering.

Infrared data access (IrDA) is supported by a fast FIR core and part of I/O block 140.

Block 140 also contains general purpose input/output which can support items such as CCD/CMOS imager module control for tuning AGC gain and electronic shutter, RTC control, battery power detection which can generate inner interrupt to the ARM for appropriate system response, camera lens motor control for focus and zoom, a user keypad input, LED indicators, flash light control, and power management control.

iMX programming

DSP 122 instructs iMX 124 to perform tasks by sending iMX commands. These commands can be complex jto understand and contain many parameters that are fixed in the inner loops. The dieal model is to provide separate command building and command-transfer routines to the DSP progarmmer, so that the commands can be pre-constructed outside the loop, and transferred to iMX as generaic data memory moves Inside the loop. Commonly used iMX commands are prepackaged in C code to ease the programming.

ARM/DSP task allocation

ARM 130 runs an operating system such as Windows CE, controls low frequency, synchronous input/output (such as to a compact flash card (CFC), and controls user interactions which also are slow and all the peripheral modules control preview engine, burst mode compression, TV encoder, CCD controller, USB, CF, IrDA, etc.

DSP 122 runs an operating system such as SPOX, controls all real-time functions (auto focus, auto exposure, auto white balance), real-time input/output (audio IO, modem IO), real-time applications (e.g., audio player), computational expensive signal processing tasks (image pipeline, JPEG 2000, image stitching).

Pin description of integrated circuit chip

The preferred embodiment pins are as follows

CCD SENSOR

Pin Count : 16

1.C_PCLK	(I)	Pixel clock
2.C_VSYNC	(I/O)	Vertical sync
3.C_HSYNC	(I/O)	Horizontal sync
4.C_FIELD	(I/O)	Field indicator
5. C_WEN	(I)	CCDC write enable
6:17.C_DATA	(I)	Image data 12Bit

SDRAM Interface

Pin Count : 58

1.SDR_CLK	(O)	Master clock
2.SDR_CKE	(O)	Clock enable
3.SDR_WE	(O)	Write enable
4.SDR_CAS	(O)	Column address strobe
5.SDR_RAS	(O)	Raw address strobe
6.SDR_CS0	(O)	Support 2pc of RAM
7.SDR_CS1	(O)	Support 4pc of RAM
8:39.DQ[31:0]	(I/O)	Data bus
40:54.SDR_A[14:0]	(O)	Address bus
55.SDR_DQMHH	(O)	DQMH for DQ[31:24]
56.SDR_DQMHL	(O)	DQMH for DQ[23:16]
57.SDR_DQMLH	(O)	DQMH for DQ[15:8]
58.SDR_DQMLL	(O)	DQMH for DQ[7:0]

ARM BUS

Pin Count : 39

1:23.ARM_A[22:0]	(O)	Address bus
24:39.ARM_D[15:0]	(O)	Data bus

Audio Interface

Pin Count : 6

1.DSP_BDX	(O)	Serial port transmit
2.DSP_BCLKX	(I/O)	Transmit clock
3.DSP_BFSX	(I/O)	Frame syncronization pulse
4.DSP_BDR	(I)	Serial data receive
5.DSP_BCLKR	(I)	Receive clock
6.DSP_BFSR	(I)	Frame synchronization pulse receive

External Flash Interface

Pin Count : 5

1.FLSH_WE	(O)	Write enable
2.FLSH_CE	(O)	Chip select

3.FLSH_OE	(O)	Output enable
4.FLSH_SIZE	(I)	8Bit/16Bit select
5.FLSH_BSY	(I)	Busy input

USB(T.B.D)

Pin Count : 10

1.M48XO	(O)	48MHz clock output
2.M48XI	(I)	48MHz clock input
3.USB_DP	(I/O)	Differential data+
4.USB_DM	(I/O)	Differential data-
5.ATTACH	(I)	Attach detect

UART

Pin Count : 5

1.RXD	(I)	UART RX
2.TXD	(O)	UART TX
3.ERXD	(I)	UART Rx for external CPU
4.ETXD	(O)	UART Tx for external CPU
5.SIFDO	(O)	Serial I/F data output

IrDA

Pin Count : 2

1.IRXD	(I)	IrDA RX
2.ITXD	(O)	IrDA TX

Compact Flash

Pin Count : 9

1.CFE1	(O)	Card enable#1
2.CFE2	(O)	Card enable#2
3.IOIS16	(O)	I/O select
4.STSCHG	(I/O)	Status changed
5.CFWAIT	(I)	Wait signal input
6.CFRST	(O)	Reset
7.CFD1	(I)	Card Detect pin#1
8.CFD2	(I)	Card Detect pin#2
9.CFRDY	(I)	Ready

TV/RGB DAC Analog output

Pin Count : 27

1.IREF(R)	(I)	R-ch Current reference control
2.DAOUT(R)	(O)	Analog output R-ch
3.GNDA		Analog GND
4.VCCA		Analog VCC

5.BIAS		(I) Phase compensation cap.R-ch
6.VREF		(I) RGB common reference voltage
7. IREF(G)	(I)	G-ch Current reference control
8. DAOUT(G)		(O) Analog output G-ch
9. GNDA		Analog GND
10. VCCA		Analog VCC
11. BIAS	(I)	Phase compensation cap.G-ch
12. IREF(B)	(I)	B-ch Current reference control
13. DAOUT(B)	(O)	Analog output B-ch
14. GNDA		Analog GND
15. VCCA		Analog VCC
16. BIAS	(I)	Phase compensation cap.B-ch
17. IREF(C)	(I)	Composite Current reference control
18. DAOUT(C)	(O)	Analog output Composite
19. GNDA		Analog GND
20. VCCA		Analog VCC
21. VREF		(I) Composite reference voltage
22. BIAS	(I)	Phase compensation cap.composite
23.DVCC		Digital VCC for DAC
24.DGND		Digital GND for DAC
25.HSYNC	(O)	H-sync output for RGB output
26.VCSYNC	(O)	V-sync / Composite-sync(select by register)

GIO

Pin Count : 32 [31:0]

1:32.GIO	(I/O)	General Purpose I/O

Micellnaeous

Pin Count : 15

1.RESET	(I)	Power on reset
2.M27XI		(I) 27MHz input
3.M27XO	(O)	27MHz output
4.TCK	(I)	JTAG clock
5.TDI	(I)	JTAG data input
6.TDO	(O)	JTAG data output
7.TMS	(I)	JTAG test mode select
8.TRST		(I) JTAG test reset
9.EMU0		(I/O) Emulator interrupt 0 pin
10.EMU1		(I/O) Emulator interrupt 1 pin
11.TEST0		(I) Test input 0
12.TEST1	(I)	Test input 1
13.SCAN		(I) Test input
14.TESTSL0	(I)	Test mode select 0
15.TESTSL1	(I)	Test mode select 1

TOTAL PIN COUNT

CCD SENSOR		17
SDRAM I/F	58	
ARM BUS	39	
Audio I/F	6	
Flash memory I/F	5	
USB	5	
UART	5	
IrDA	2	
Compact Flash I/F	9	
4DAC	26	
GIO	32	
Micellnaeous	15	
Sub Total	219 pins	
Power :	37 pins (14%)	
TOTAL	: 256 pins	

Audio player

Portable digital audio players are expected to be one of the most popular consumer products. Currently the MP-3 player based on MPEG-1 Layer 3 ausio compression standard is growing rapidly in portable audio market while MPEG-2 AAC and Doby AC-3 are alternative digital audio coding formats to be considered as emerging standards. Thus the preferred embodiments's programmability permits inclusion of digital audio player functions. The audio can be input via flash memory, PC, etc. and the decoded can be output on the serial port. The decoding program can be loaded from flash memory, ROM, etc.

CLAIMS

What is claimed is:

1. An integrated circuit for a digital still camera, comprising:

(a) a first programmable processor programmed to run control functions, said first processor coupled to a user interface, a controller for memory, and a controller for image acquisition;

(b) a second programmable processor programmed to run image processing and compression functions, said second processor coupled to said first processor; and

(a) a third processor coupled to said second processor, said third processor including at least four parallel multiply and accumulate units.

2. A method of resizing a digital image, comprising the steps of:

(a) providing an M1 by M2 pixel image where M1 and M2 are positive integers;

(b) filtering a row of M1 pixels of the image with a set of filter coefficients which depends upon pixel location to yield M1 filtered outputs;

(c) discarding M1-N1 of the filtered outputs of step (b);

(d) repeating steps (b) and (c) for all other rows of the M1 by M2 image to yield a N1 by M2 image;

(e) filtering a column of M2 pixels of the N1 by M2 image with a set of filter coefficients which depends upon pixel location to yield M2 filtered outputs;

(f) discarding M2-N2 of the filtered outputs of step (e); and

(g) repeating steps (e) and (f) for all other rows of the N1 by M2 image to yield a N1 by N2 image.

3. A method of color-filtered array interpolation, comprising the steps of:

interpolate a first color subarray to yield a first color array;

high-pass filter the first color array from the prior step to yield a high pass first color array;

interpolate a second color subarray to yield a second color array;

weight and add the high-pass first color array to the second color array to yield a modified second color array;

interpolate a third color subarray to yield a third color array;

weight and add the high-pass first color array to the third color array to yield a modified third color array;

defining a three-color image with the first color array, the modified second color array, and the modified third color array.

4. A method of tone-scaling an image, comprising the steps of:

(a) providing an M by N pixel image where M and N are positive integers and each pixel has an intensity in the range from 0 to K;

(b) determining a scaled cumulative distribution function T(X) for X in the range 0 to K and with values in the range 0 to K, T(X) equals K multiplied by the number of pixels with intensity less than or equal to X and then divided by MN;

(c) selecting a parameter A in the range 0 to 1;

(d) adjusting an intensity Z of a pixel of step (a) to AT(Z)+(1-A)Z.

5. A method of interpolation for a Bayer color-filtered array, comprising the steps of:

interpolate the green subarray to form a tentative green array;

clamp the interpolated pixel values of the tentative green array to lie in the range of the middle two values of the four neighboring values of the green subarray to form a green array; and

interpolate the red and blue subarrays.

6. A method of interpolation for a Bayer color-filtered array image, comprising the steps of:

interpolate the green subarray to form a green array;

diagonally interpolate the red subarray to form a second red subarray;

horizontally and vertically interpolate the second red subarray to form a red array;

diagonally interpolate the blue subarray to form a second blue subarray;

horizontally and vertically interpolate the second blue subarray to form a blue array;

wherein the diagonal interpolation includes weightings by the values of the corresponding pixels in the green array;

wherein the horizontal and vertical interpolation includes weightings by the values of the corresponding pixels in the green array; and

the green array, red array, and blue array define a color image.

7. A method of white balancing for a color image sensor, comprising the steps of:

(a) providing output green, red, and blue pixel intensities for a color image sensor illuminated with white light;

(b) determining the ratio of the green pixel intensities to the blue pixel intensities for the outputs of step (a);

(c) providing an adjustment for blue pixel intensities as multiplication by the ratio of step (b) plus addition of an offset which depends upon the pixel intensities in step (a);

(d) determining the ratio of the green pixel intensities to the red pixel intensities for the outputs of step (a); and

(c) providing an adjustment for red pixel intensities as multiplication by the ratio of step (b) plus addition of an offset which depends upon the pixel intensities in step (a).

8. A method of video decoding and display, comprising the steps of:

(a) providing a sequence of encoded video frames, a sequence of video frame boundary times, and storage locations for decoded ones of the frames;

(b) when the mth frame of the sequence has been decoded, next decode the m'th frame of the sequence with m' determined as TP_n divided by DeltaT

where DeltaT is the time interval between successive ones of the video frame boundary times of step (a) and TP_n is a presentation time determined as the greater of (i) the presentation time TP_{n-1} of the mth frame plus DeltaT and (ii) the starting time Ts for decoding the frame to be presented next after the mth frame plus an estimated time to decode a frame, the sum rounded up to a multiple of DeltaT, wherein the starting time Ts is the greater of (i) current time and (ii) the time a storage location to contained the decoded m'th frame will be available due to a prior decoded frame being displayed;

(c) displaying a decoded frame;

(d) repeating steps (b) and (c) with incremented indices.

9. A method of interpolation for a complementary-color-filtered array image, comprising the steps of:

(a) provide a complementary-color-filtered array of pixel values with yellow pixel values Ye on a first subarray, cyan pixel values Cy on a second subarray, magenta pixel values Mg on a third subarray, and green pixel values G on a fourth subarray;

(b) interpolating the yellow pixel values to all pixels in the array;

(c) interpolating the cyan pixel values to all pixels in the

(d) interpolating the magenta pixel values to all pixels in the array;

(e) interpolating the green pixel values to all pixels in the array;

(f) adjusting the pixel yellow value by subtracting quantity (Ye + Cy – 2*G – Mg)/4 from the pixel yellow value from step (b) where Ye is the pixel yellow value from step (b), Cy is the pixel cyan value from step (c), Mg is the pixel magenta value from step (d), and G is the pixel green value from step (e);

(g) adjusting the pixel cyan value by subtracting quantity (Ye + Cy – 2*G – Mg)/4 from the pixel cyan value from step (c) where Ye is the pixel yellow value from step (b), Cy is the pixel cyan value from step (c), Mg is the pixel magenta value from step (d), and G is the pixel green value from step (e);

(h) adjusting the pixel magenta value by adding quantity (Ye + Cy – 2*G – Mg)/4 to the pixel magenta value from step (d) where Ye is the pixel yellow value

from step (b), Cy is the pixel cyan value from step (c), Mg is the pixel magenta value from step (d), and G is the pixel green value from step (e); and

(i) adjusting the pixel green value by adding quantity (Ye + Cy – 2*G – Mg)/8 to the pixel green value from step (e) where Ye is the pixel yellow value from step (b), Cy is the pixel cyan value from step (c), Mg is the pixel magenta value from step (d), and G is the pixel green value from step (e).

10. A method of interpolation for a color-filtered array image, comprising the steps of:

(a) provide a color-filtered array of pixel values with red pixel value R(j,k) for the pixel located at the jth row and kth column when j and k are even integers, blue pixel value B(j,k) for pixel located at jth row and kth column when j and k are odd integers, and green pixel value G(j,k) for the pixel located at the jth row and kth column when one of j and k is an even integer and the other an odd integer;

(b) horizontally interpolate rows of pixel values when j is even to yield rows of red values R(j,k) and green values G(j,k) for all k, and when j is odd to yield green values G(j,k) and blue values B(j,k) for all k;

(c) when j is an even integer vertically interpolate the horizontal interpolated red rows from step (b) to yield the red values R(j-1,k) and when j is an odd integer vertically interpolate the horizontal interpolated blue to yield the blue values B(j-1,k);

(d) vertically filter the horizontal interpolated green pixel rows from step (b) to yield filtered green output values G"(j,k);

(e) define the difference values D(j,k) between the green values G(j,k) and the filtered green values G"(j,k);

(f) when j is even subtract D(j,k) from the red values R(j,k) to yield red output values R"(j,k) and when m is odd add D(j,k) to the red values R(j,k) to yield red output values R"(j,k); and

(g) when m is even add D(j,k) to the blue values B(j,k) to yield blue output values B"(j,k) and when m is odd subtract D(j,k) from the blue values B(j,k) to yield blue output values B"(j,k).

11. A method of playback buffering, comprising:

(a) providing first and second buffers, each of the buffers with a full/not-full indicator, and a position indicator indicating a position within the first and second buffers;

(b) filling the first buffer with encoded bits when the full/not-full indicator for the first buffer indicates not-full and the position indicator indicates a position in the second buffer;

(c) filling the second buffer with encoded bits when the full/not-full indicator for the second buffer indicates not-full and the position indicator indicates a position in the first buffer;

(d) decoding a group of bits in the first and second buffers starting at the position indicated by the position indicator and treating the first and second buffers as forming a single circular buffer;

(e) updating the full/not-full indicators according to the bits removed by step (d); and

(f) repeating steps (b)-(e).

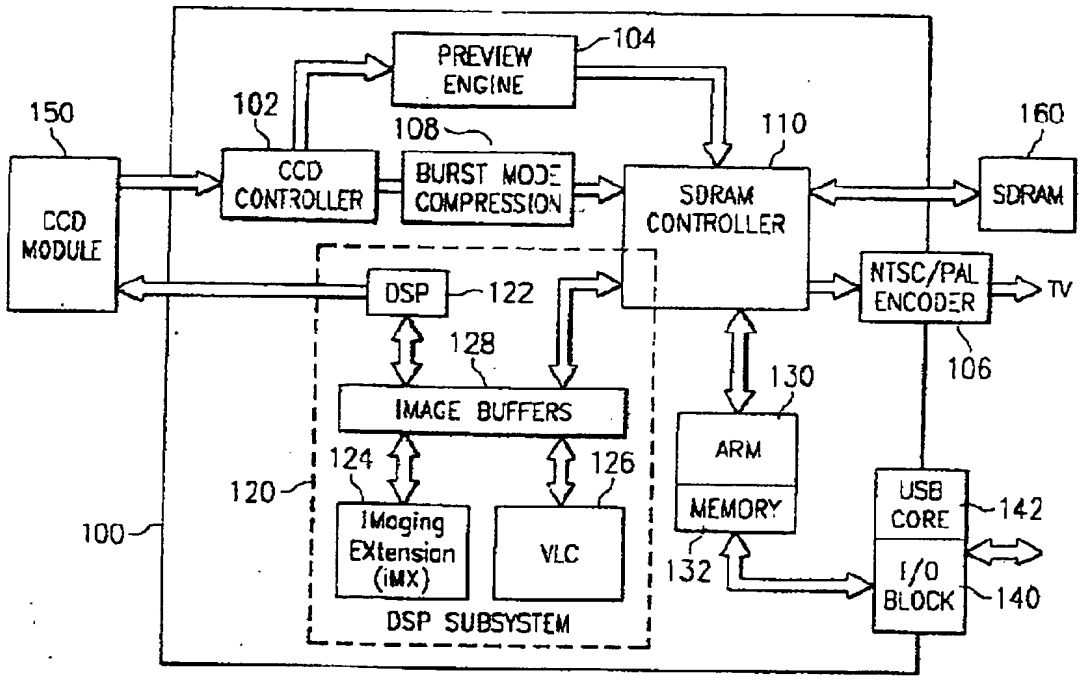

PREVIEW ENGINE
104
150
102
108
110
160
CCD MODULE
CCD CONTROLLER
BURST MODE COMPRESSION
SDRAM CONTROLLER
SDRAM
DSP
122
NTSC/PAL ENCODER
TV
128
106
IMAGE BUFFERS
130
ARM
MEMORY
124
126
120
USB CORE
142
Imaging EXtension (iMX)
VLC
132
I/O BLOCK
140
100
DSP SUBSYSTEM

FIG. 1a

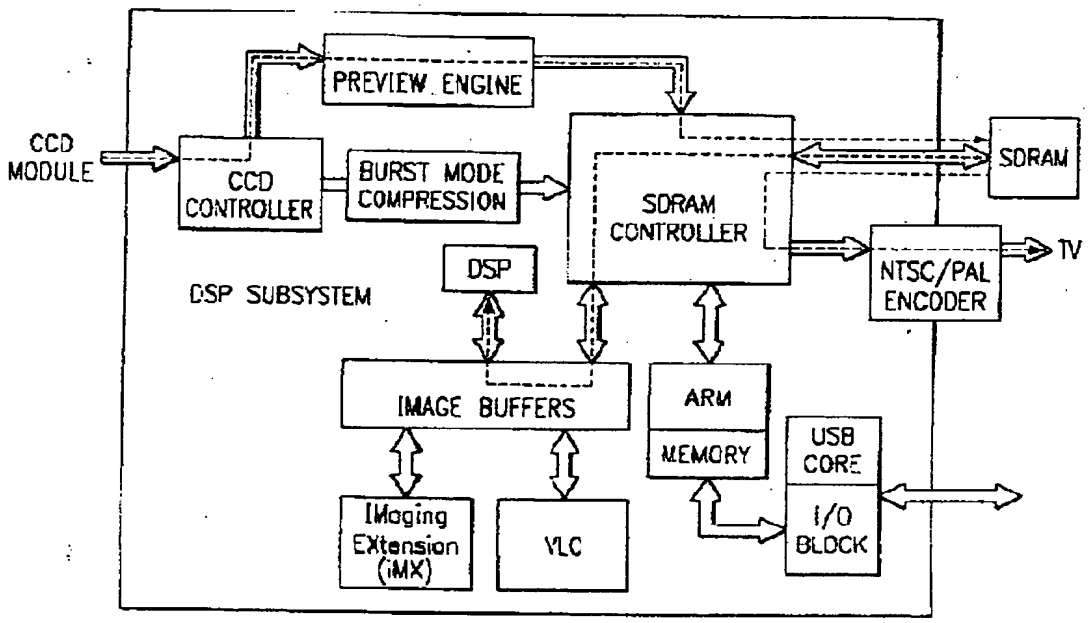

PREVIEW ENGINE
CCD MODULE
CCD CONTROLLER
BURST MODE COMPRESSION
SDRAM CONTROLLER
SDRAM
TV
NTSC/PAL ENCODER
DSP SUBSYSTEM
DSP
IMAGE BUFFERS
ARM
MEMORY
USB CORE
I/O BLOCK
Imaging EXtension (iMX)
VLC

FIG. 2

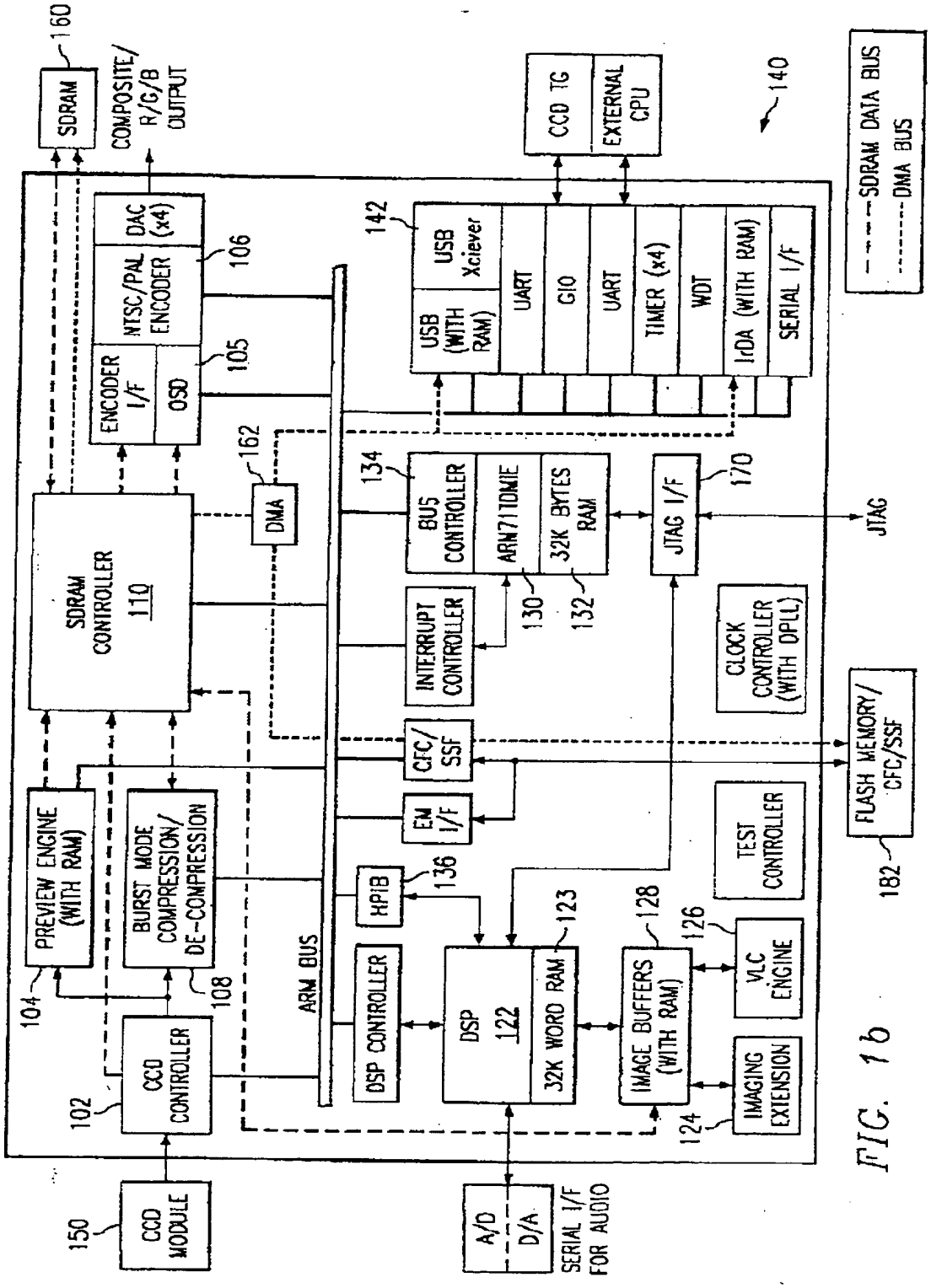

SDRAM
160
COMPOSITE/ R/G/B OUTPUT
DAC (x4)
NTSC/PAL ENCODER
106
ENCODER I/F
OSD
105
SDRAM CONTROLLER 110
DMA
162
PREVIEW ENGINE (WITH RAM)
104
BURST MODE COMPRESSION/ DE-COMPRESSION
108
CCD CONTROLLER
102
CCD MODULE
150
ARM BUS
DSP CONTROLLER
HPIB
136
DSP 122
32K WORD RAM
123
IMAGE BUFFERS (WITH RAM)
128
126
VLC ENGINE
IMAGING EXTENSION
124
A/D D/A
SERIAL I/F FOR AUDIO
CFC/ SSF
EM I/F
INTERRUPT CONTROLLER
BUS CONTROLLER
134
ARM7TDMIE
130
32K BYTES RAM
132
JTAG I/F
170
JTAG
CLOCK CONTROLLER (WITH DPLL)
TEST CONTROLLER
FLASH MEMORY/ CFC/SSF
182
CCD TG
EXTERNAL CPU
140
USB Xciever
142
USB (WITH RAM)
UART
GIO
UART
TIMER (x4)
WDT
IrDA (WITH RAM)
SERIAL I/F
SDRAM DATA BUS
DMA BUS

FIG. 1b

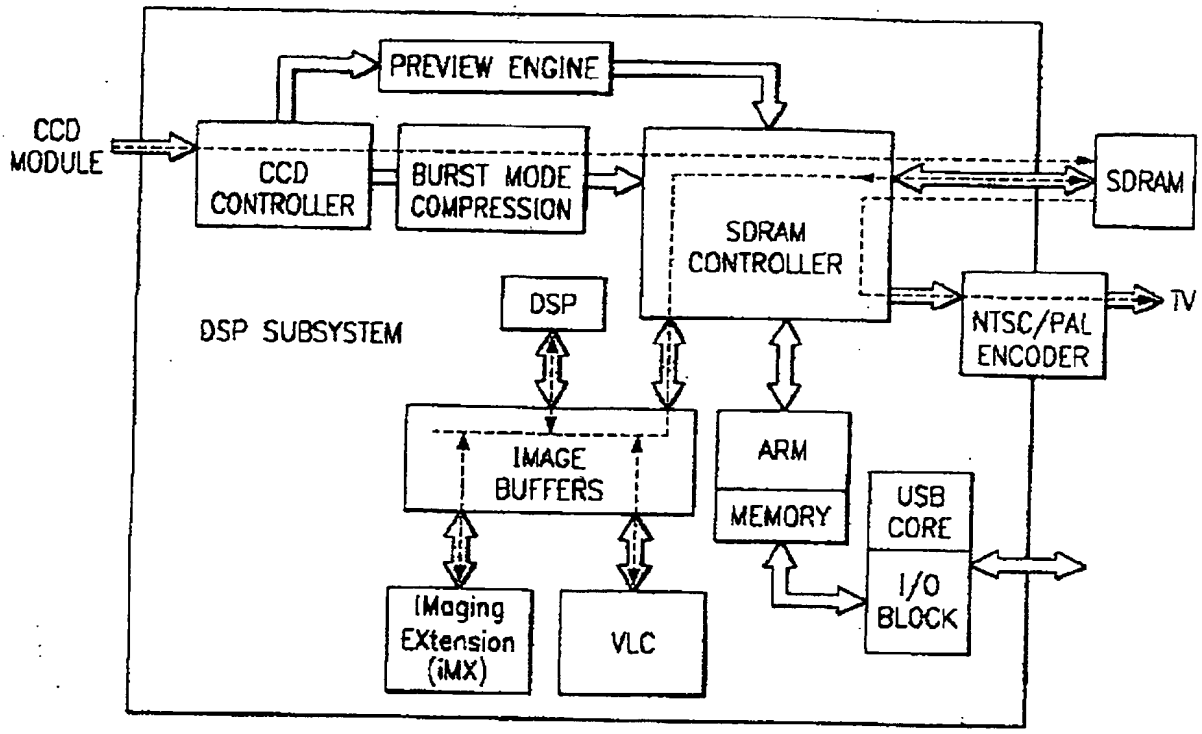

PREVIEW ENGINE
CCD MODULE
CCD CONTROLLER
BURST MODE COMPRESSION
SDRAM CONTROLLER
SDRAM
DSP
DSP SUBSYSTEM
NTSC/PAL ENCODER
TV
IMAGE BUFFERS
ARM
MEMORY
USB CORE
I/O BLOCK
IMaging EXtension (iMX)
VLC

FIG. 3a

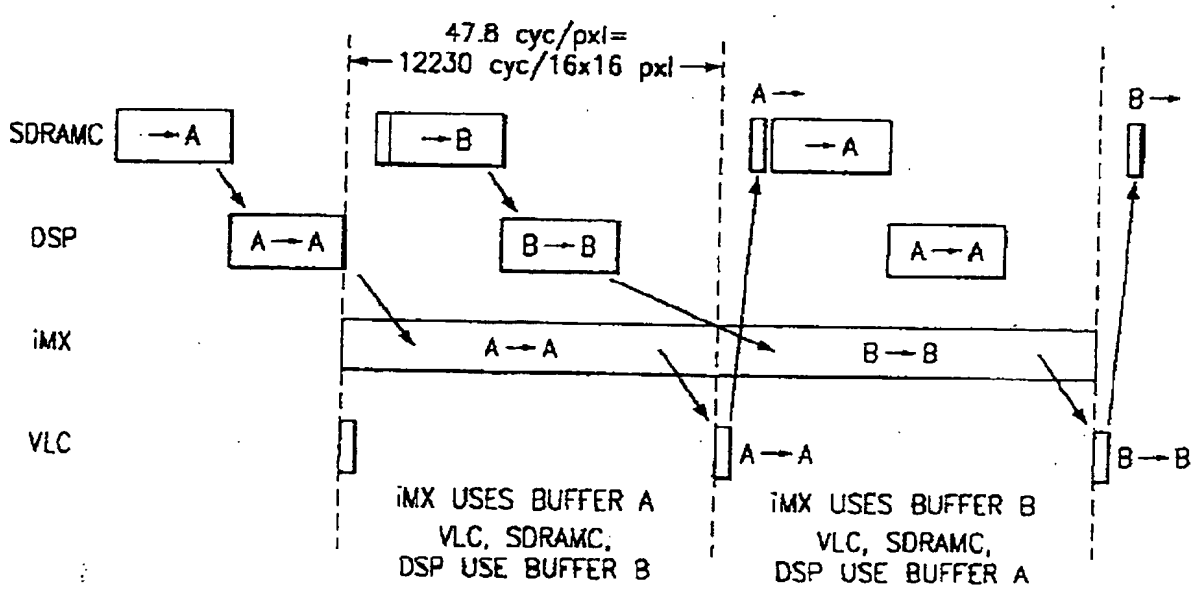

47.8 cyc/pxl=
12230 cyc/16x16 pxl
SDRAMC
DSP
iMX
VLC
A→A
B→B
iMX USES BUFFER A
VLC, SDRAMC,
DSP USE BUFFER B
iMX USES BUFFER B
VLC, SDRAMC,
DSP USE BUFFER A

FIG. 3b

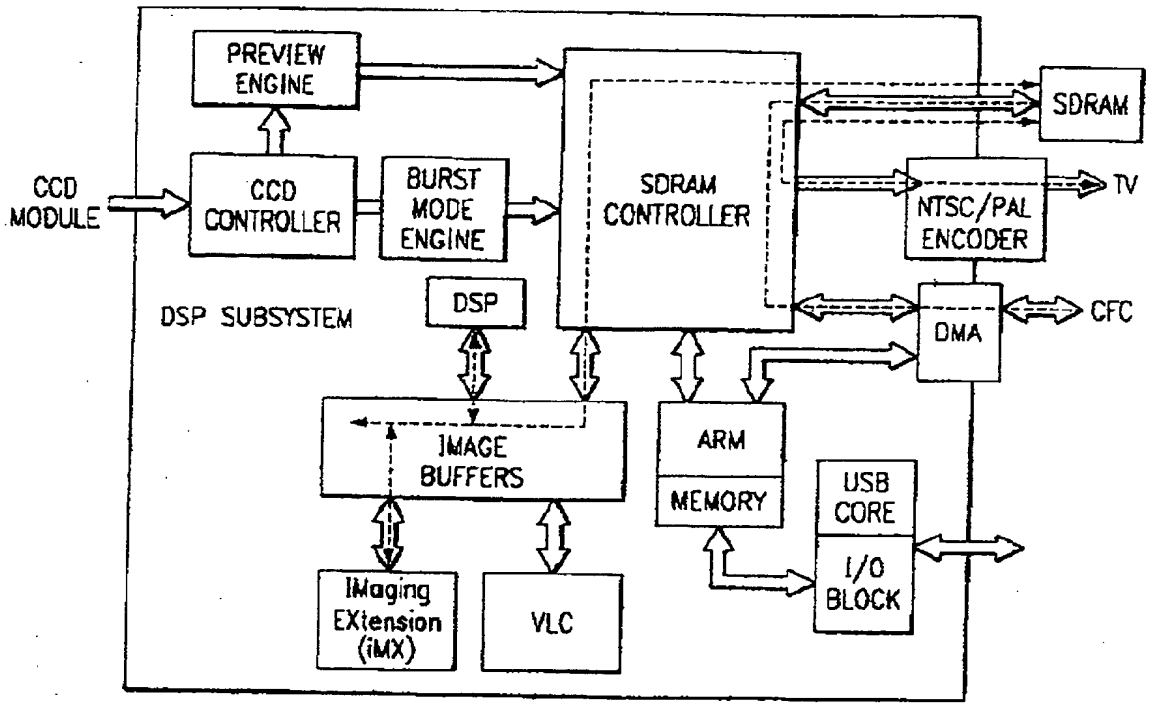

PREVIEW ENGINE
SDRAM
CCD MODULE
CCD CONTROLLER
BURST MODE ENGINE
SDRAM CONTROLLER
TV
NTSC/PAL ENCODER
DSP SUBSYSTEM
DSP
DMA
CFC
IMAGE BUFFERS
ARM
MEMORY
USB CORE
I/O BLOCK
IMaging EXtension (iMX)
VLC

FIG. 4

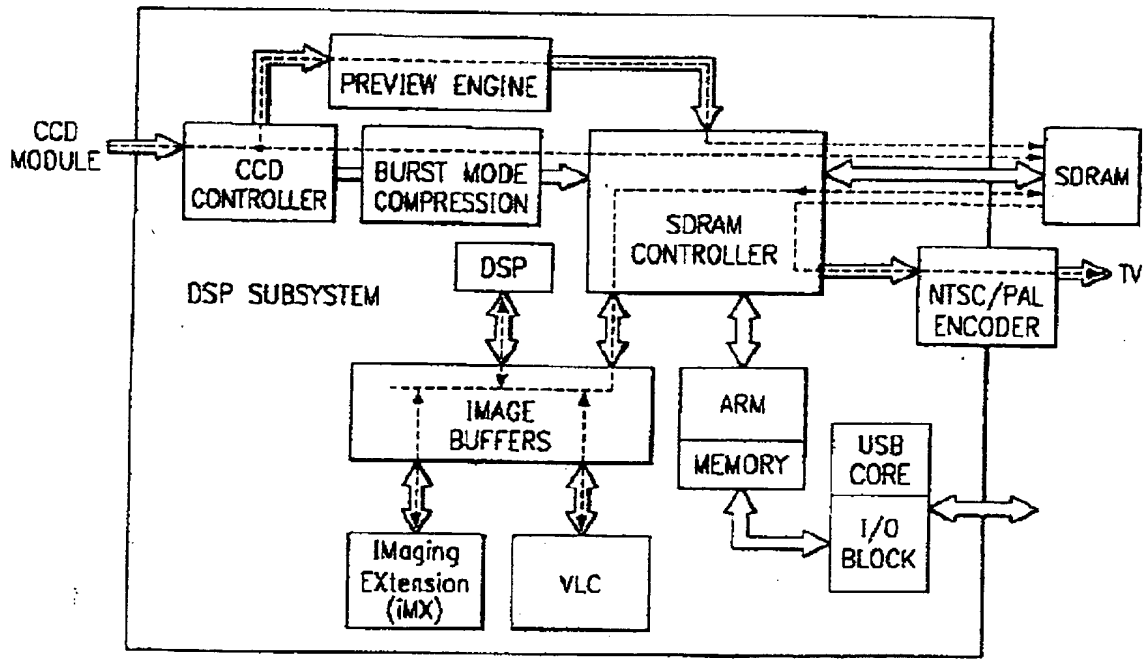

PREVIEW ENGINE
CCD MODULE
CCD CONTROLLER
BURST MODE COMPRESSION
SDRAM
SDRAM CONTROLLER
DSP
TV
DSP SUBSYSTEM
NTSC/PAL ENCODER
IMAGE BUFFERS
ARM
MEMORY
USB CORE
I/O BLOCK
IMaging EXtension (iMX)
VLC

FIG. 5

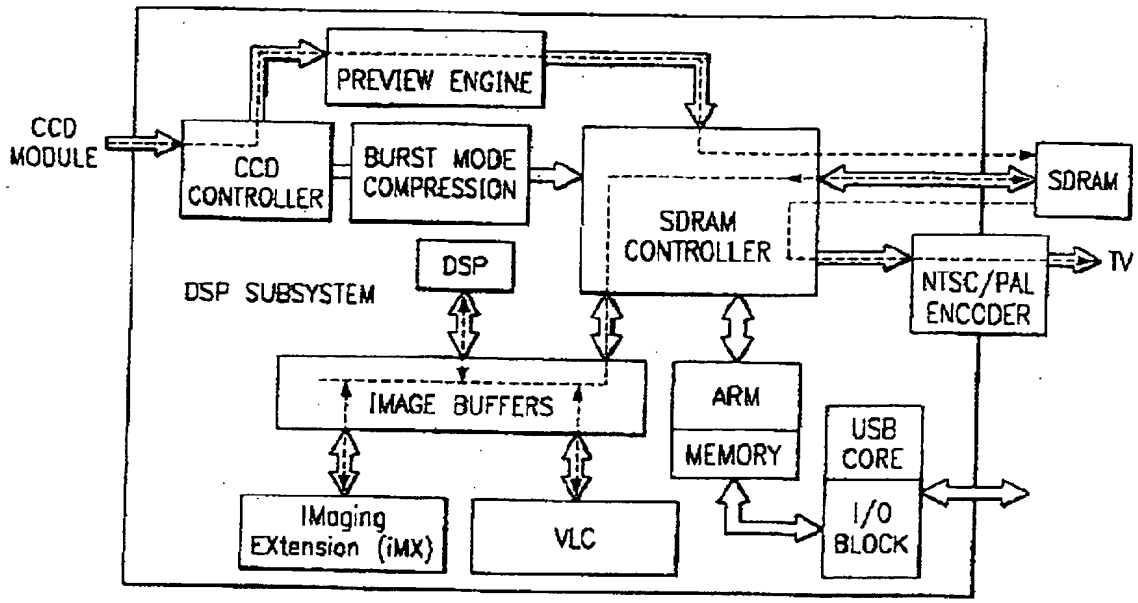


FIG. 6

R	G	R	G
G	B	G	B
R	G	R	G
G	B	G	B

FIG. 7a

Ye	Cy	Ye	Cy
G	Mg	G	Mg
Ye	Cy	Ye	Cy
G	Mg	G	Mg

FIG. 7b

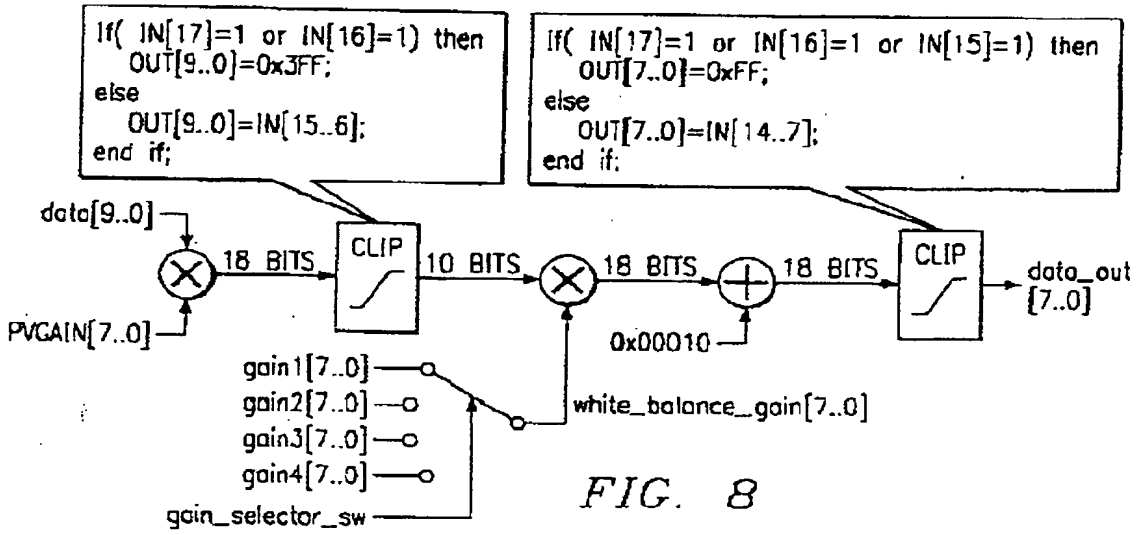


FIG. 8

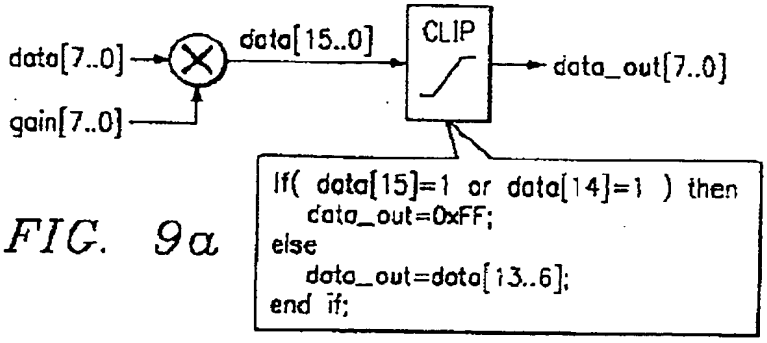


FIG. 9a

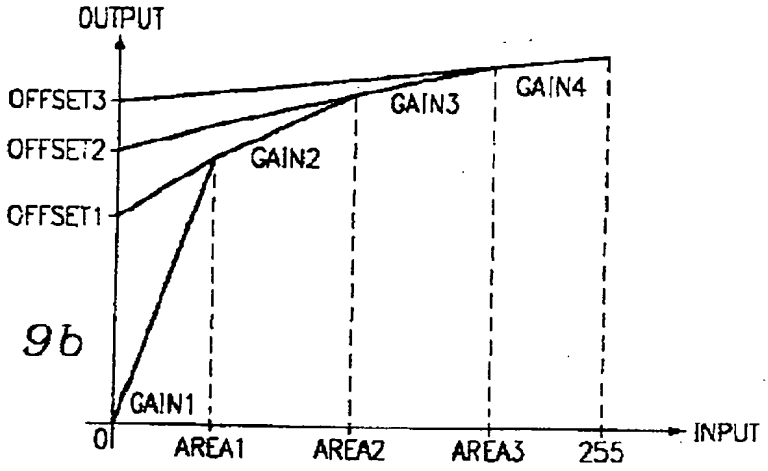


FIG. 9b

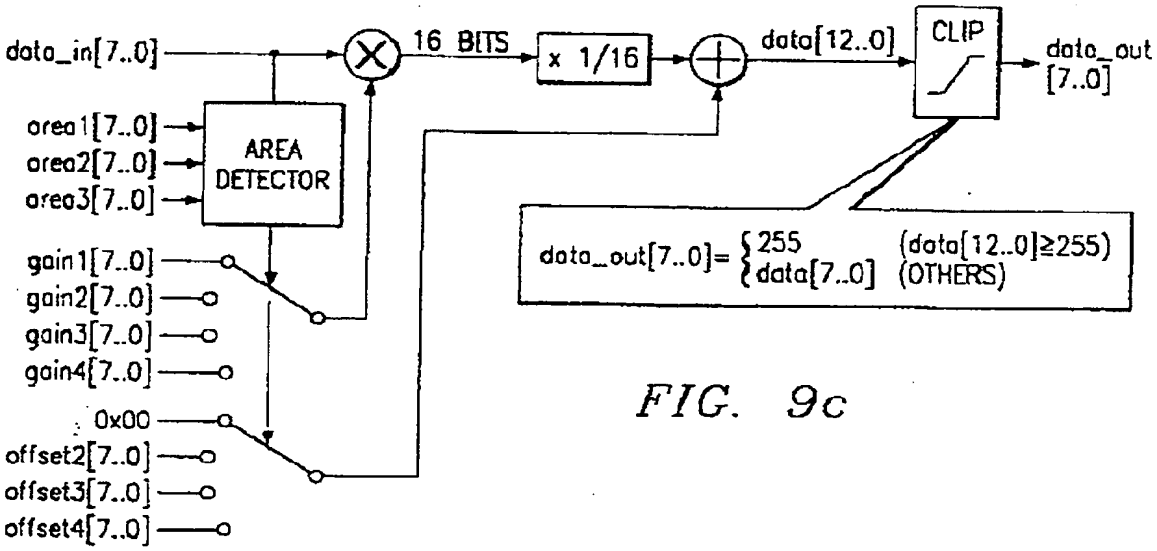


FIG. 9c

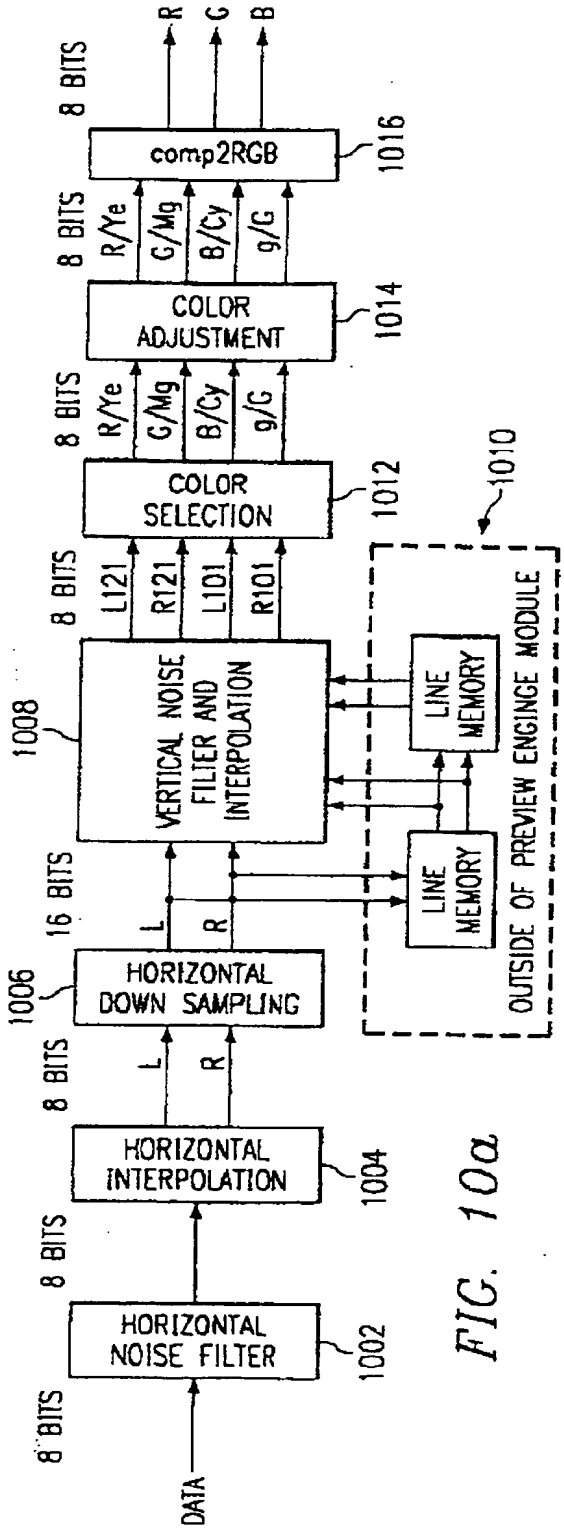


FIG. 10a

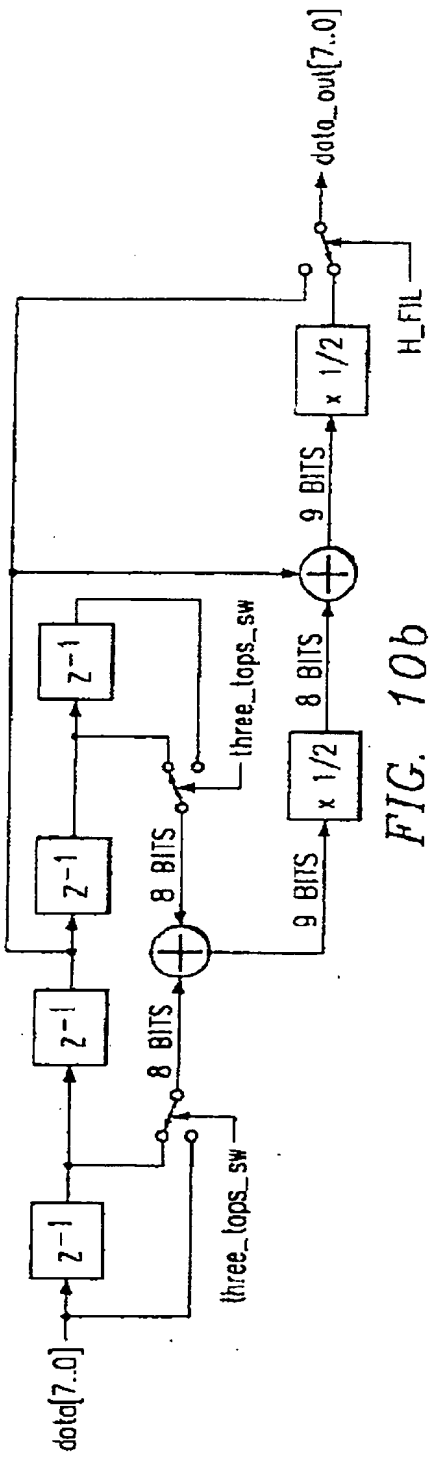


FIG. 10b

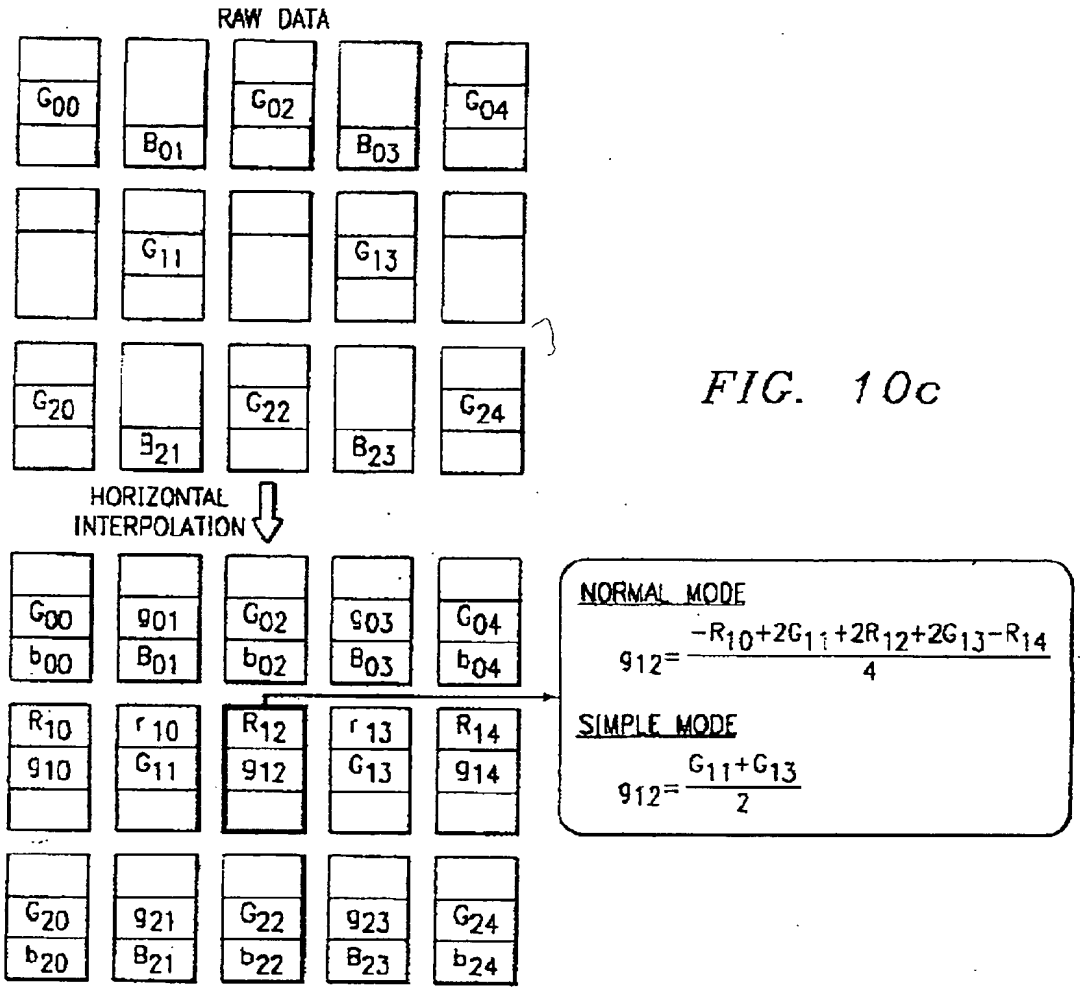


FIG. 10c

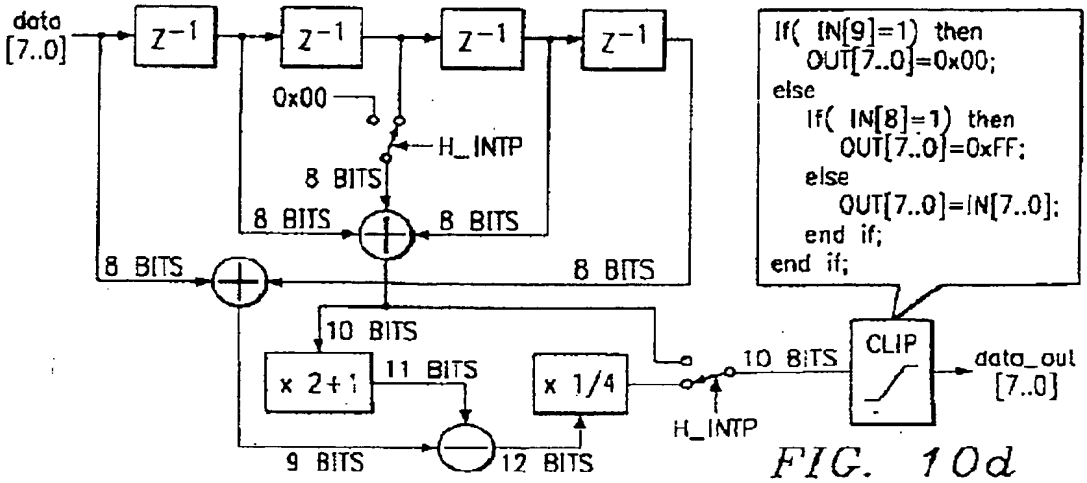


FIG. 10d

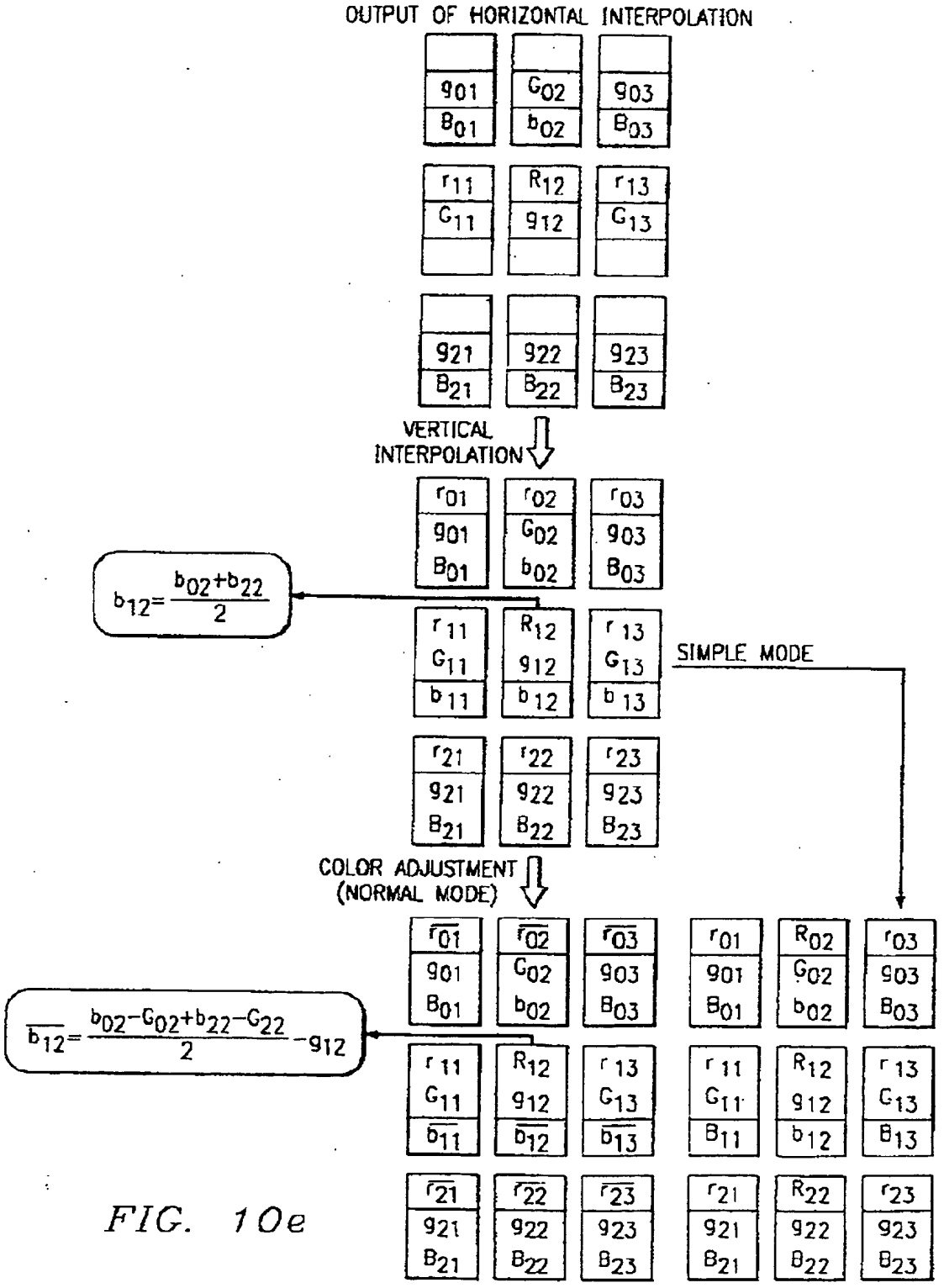

OUTPUT OF HORIZONTAL INTERPOLATION
VERTICAL INTERPOLATION
$b_{12}=\frac{b_{02}+b_{22}}{2}$
SIMPLE MODE
COLOR ADJUSTMENT (NORMAL MODE)
$\overline{b_{12}}=\frac{b_{02}-G_{02}+b_{22}-G_{22}}{2}-g_{12}$

FIG. 10e

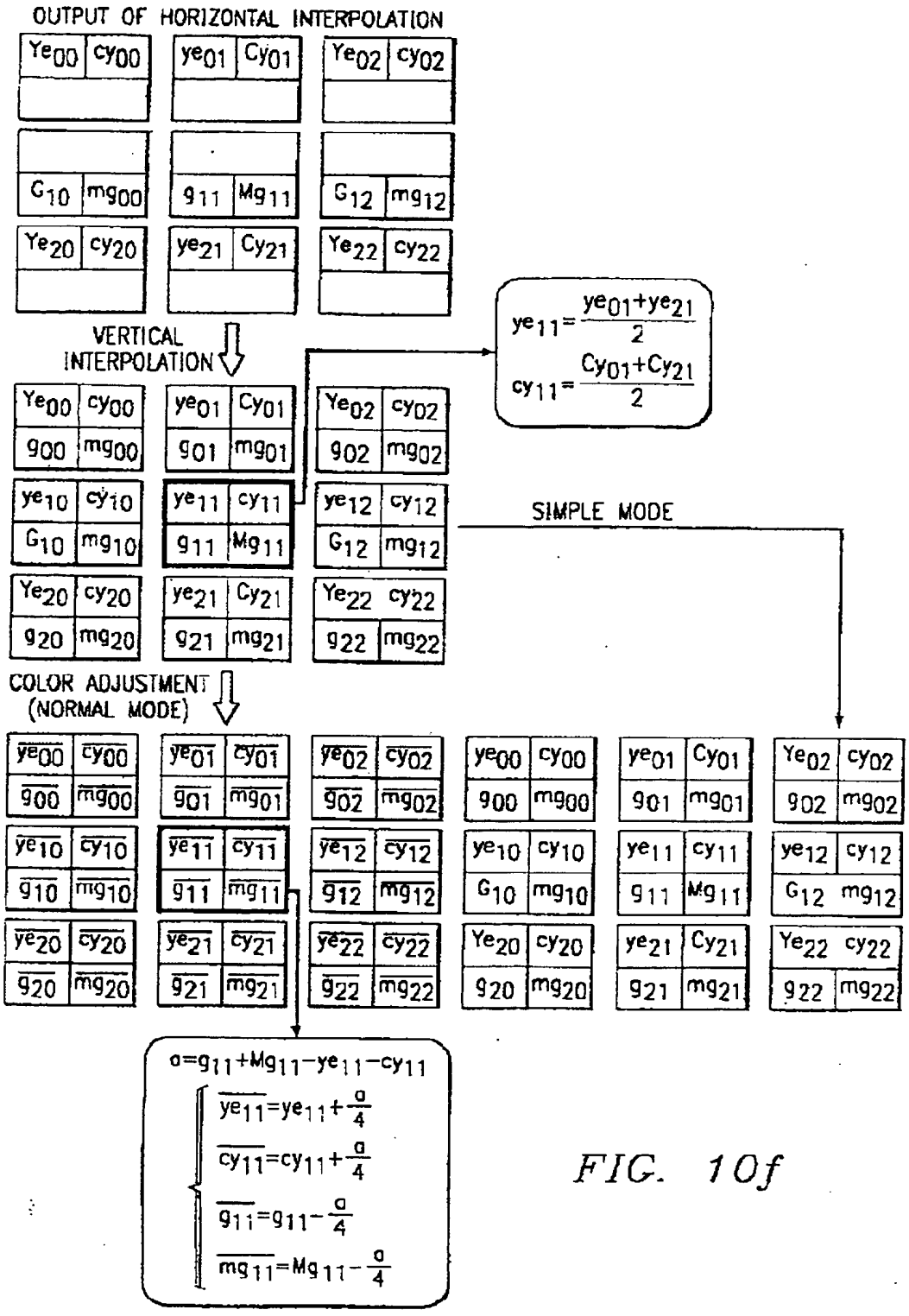
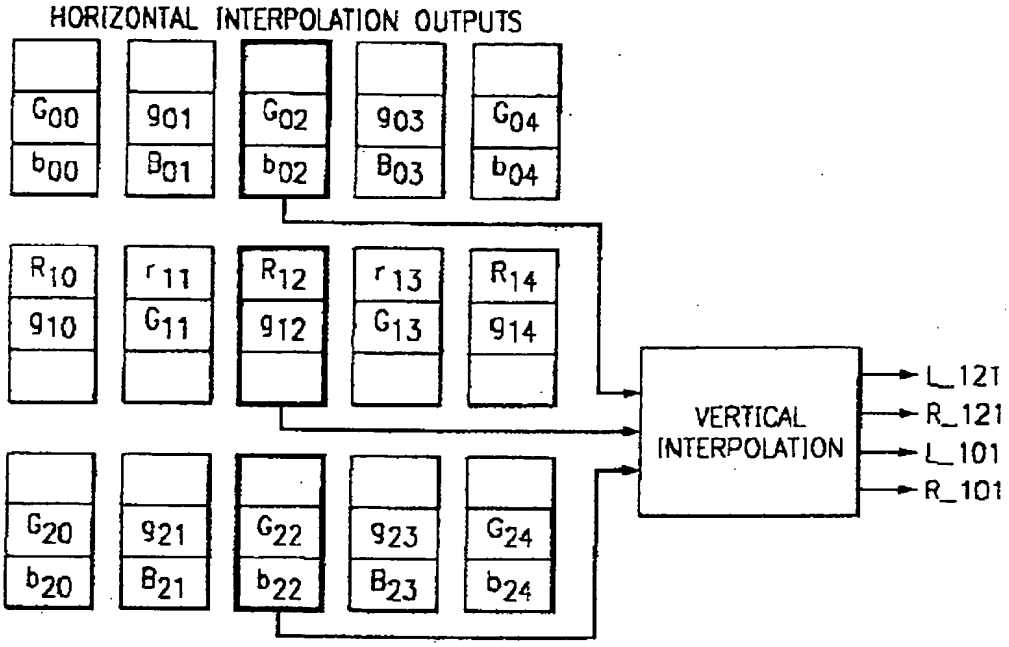

OUTPUT OF HORIZONTAL INTERPOLATION
VERTICAL INTERPOLATION
$ye_{11}=\frac{ye_{01}+ye_{21}}{2}$
$cy_{11}=\frac{Cy_{01}+Cy_{21}}{2}$
SIMPLE MODE
COLOR ADJUSTMENT (NORMAL MODE)
$a=g_{11}+Mg_{11}-ye_{11}-cy_{11}$
$\overline{ye_{11}}=ye_{11}+\frac{a}{4}$
$\overline{cy_{11}}=cy_{11}+\frac{a}{4}$
$\overline{g_{11}}=g_{11}-\frac{a}{4}$
$\overline{mg_{11}}=Mg_{11}-\frac{a}{4}$

FIG. 10f



NOISE FILTER = OFF

$$L\_121 = R_{12}$$

$$R\_121 = g_{12}$$

$$L\_101 = \frac{G_{02}+G_{22}}{2}$$

$$R\_101 = \frac{b_{02}+b_{22}}{2}$$

NOISE FILTER = ON

$$L\_121 = R_{12} - g_{12} + \frac{G_{02}+2g_{12}+G_{22}}{4}$$

$$R\_121 = \frac{G_{02}+2g_{12}+G_{22}}{4}$$

$$L\_101 = \frac{G_{02}+G_{22}}{2}$$

$$R\_101 = \frac{b_{02}+b_{22}}{2}$$

FIG. 10g

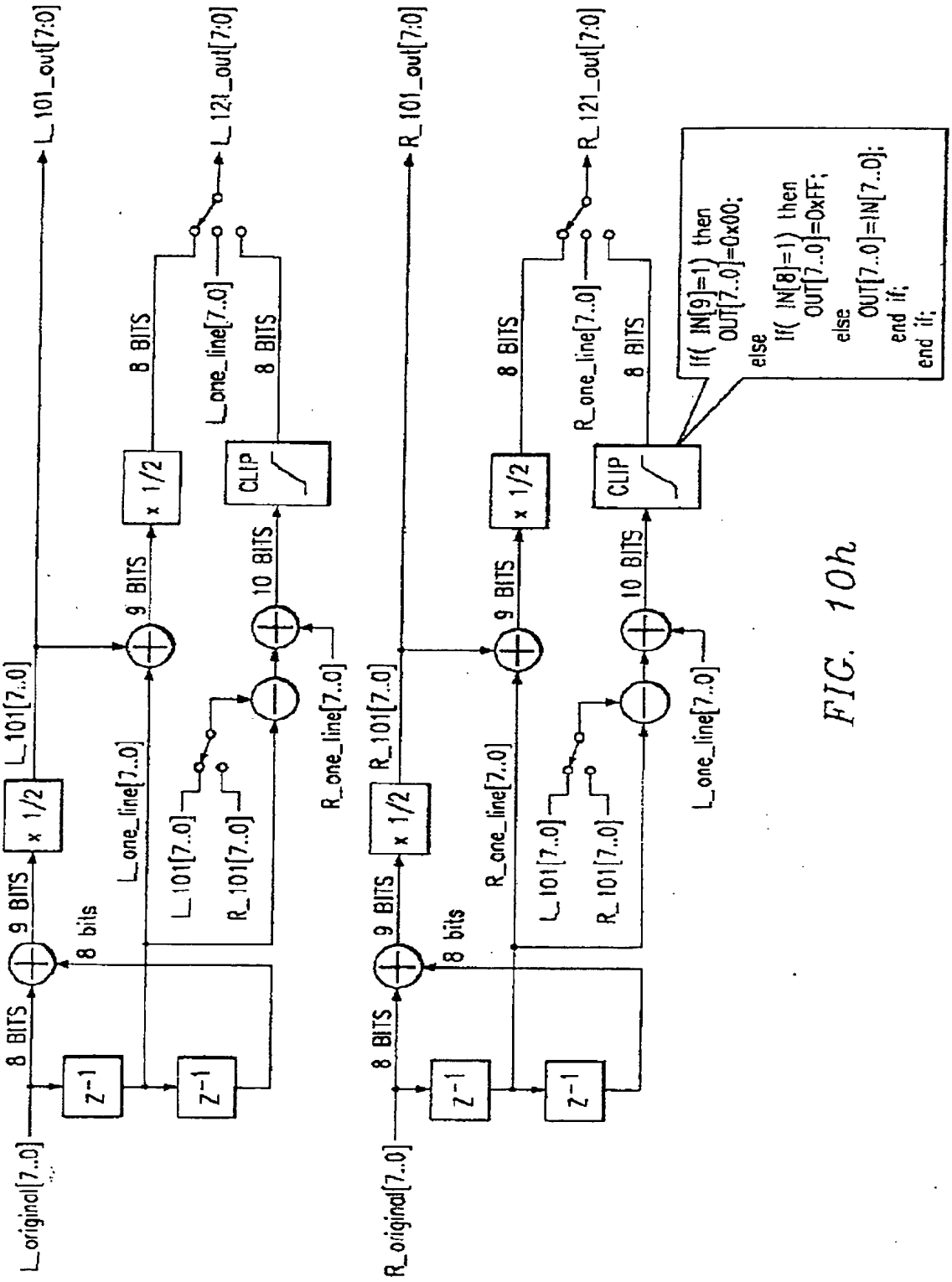
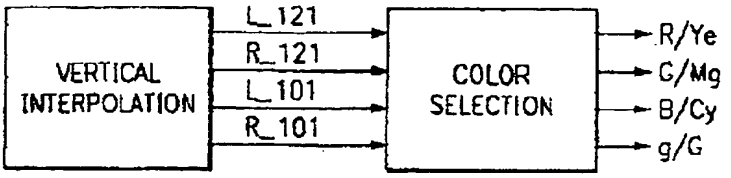

L_original[7..0]
8 BITS
Z-1
8 bits
9 BITS
x 1/2
L_101[7..0]
L_101_out[7:0]
L_one_line[7..0]
L_101[7..0]
R_101[7..0]
9 BITS
x 1/2
8 BITS
L_121_out[7:0]
L_one_line[7..0]
8 BITS
CLIP
10 BITS
R_one_line[7..0]
R_original[7..0]
R_101[7..0]
R_101_out[7:0]
R_one_line[7..0]
R_121_out[7:0]
If(IN[9]=1) then
OUT[7..0]=0x00;
else
If(IN[8]=1) then
OUT[7..0]=0xFF;
else
OUT[7..0]=IN[7..0];
end if;
end if;

FIG. 10h



NOISE FILTER = OFF

$$R/Ye = R_{12}$$
$$G/Mg = g_{12}$$
$$g/Cy = \frac{G_{02}+G_{22}}{2}$$
$$B/Cy = \frac{b_{02}+b_{22}}{2}$$

NOISE FILTER = ON

$$R/Ye = R_{12} - g_{12} + \frac{G_{02}+2g_{12}+G_{22}}{4}$$
$$G/Mg = \frac{G_{02}+2g_{12}+G_{22}}{4}$$
$$g/Cy = \frac{G_{02}+G_{22}}{2}$$
$$B/Cy = \frac{b_{02}+b_{22}}{2}$$

FIG. 10i

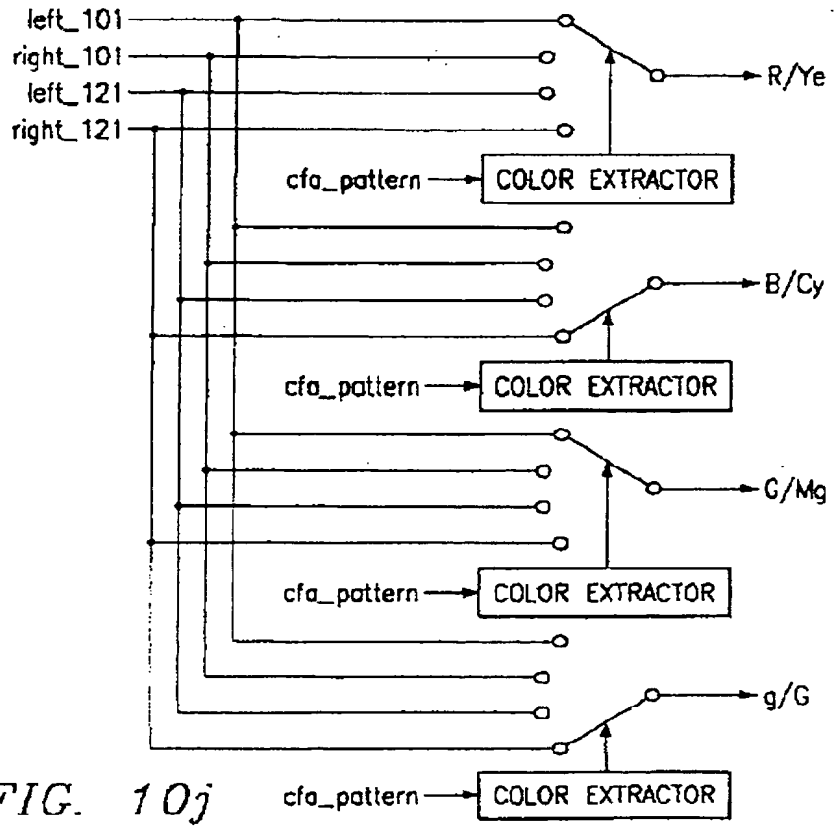


FIG. 10j

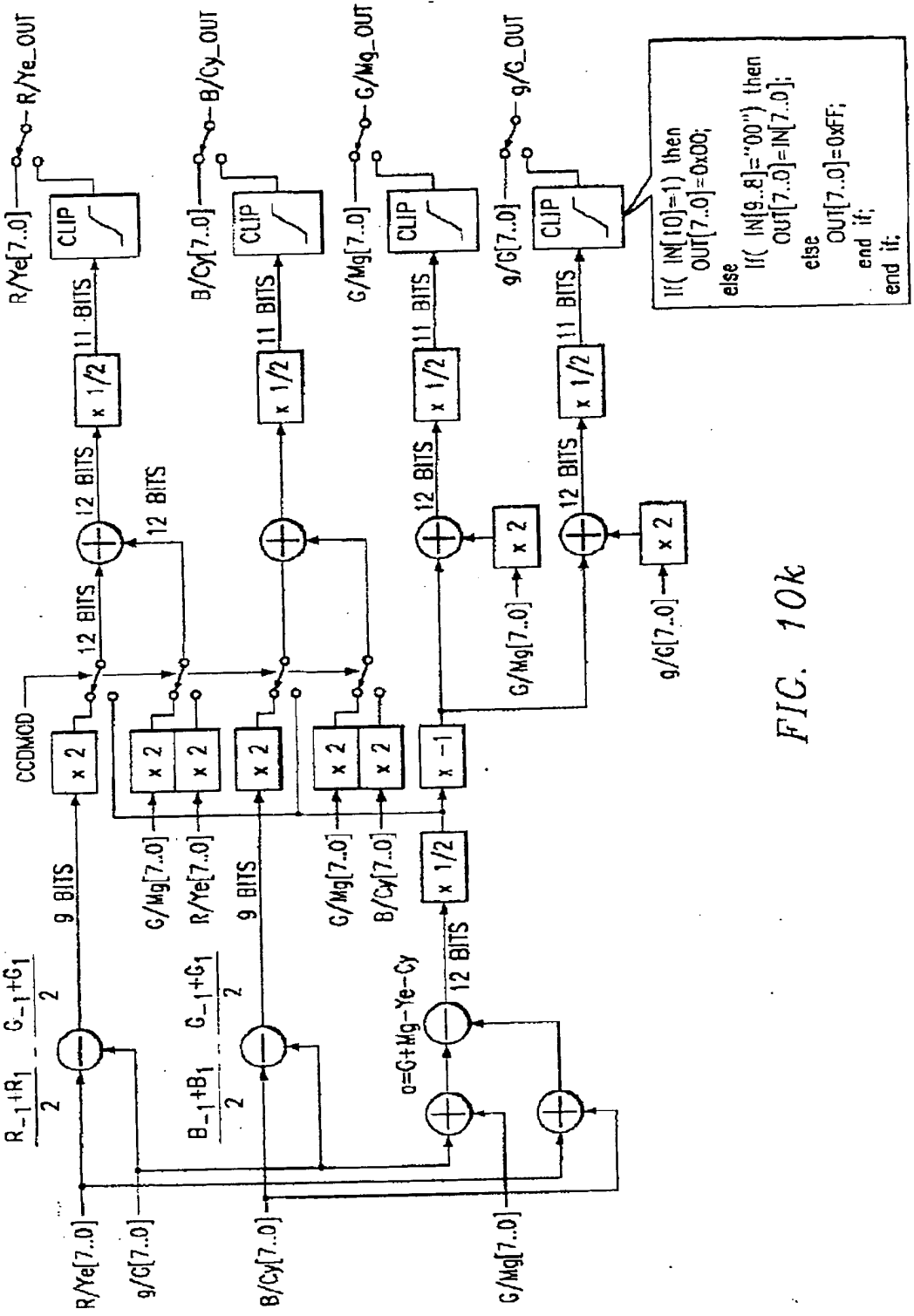

R/Ye_OUT
B/Cy_OUT
G/Mg_OUT
g/G_OUT
CLIP
11 BITS
x 1/2
12 BITS
x 2
x -1
9 BITS
CCDMOD
R/Ye[7..0]
g/G[7..0]
B/Cy[7..0]
G/Mg[7..0]
(R-1+R1)/2 - (G-1+G1)/2
(B-1+B1)/2 - (G-1+G1)/2
o=G+Mg-Ye-Cy
If(IN[10]=1) then
OUT[7..0]=0x00;
else
If(IN[9..8]="00") then
OUT[7..0]=IN[7..0];
else
OUT[7..0]=0xFF;
end if;
end if;

FIG. 10k

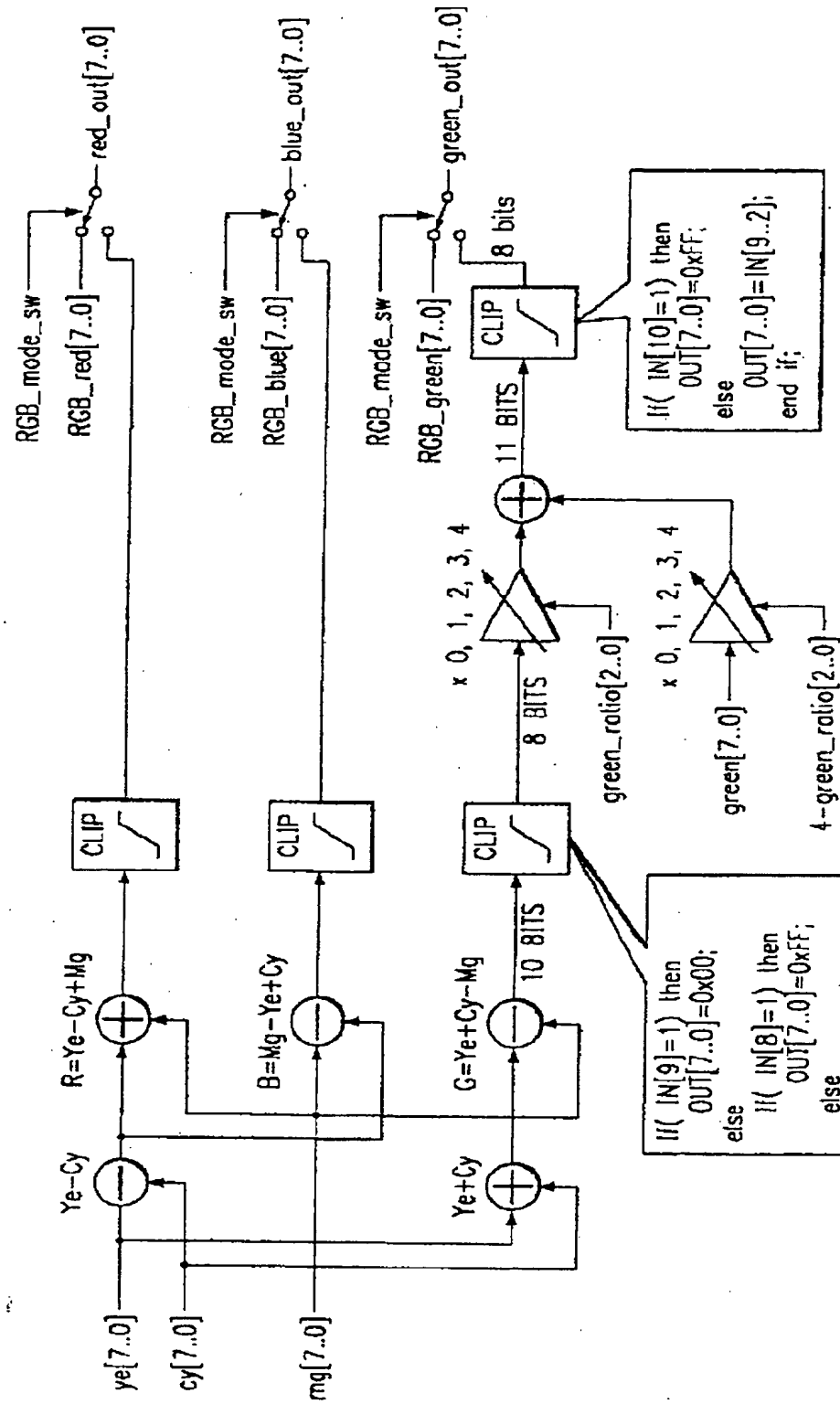

RGB_mode_sw
RGB_red[7..0]
red_out[7..0]
RGB_mode_sw
RGB_blue[7..0]
blue_out[7..0]
RGB_mode_sw
RGB_green[7..0]
green_out[7..0]
8 bits
CLIP
11 BITS
x 0, 1, 2, 3, 4
green_ratio[2..0]
8 BITS
10 BITS
R=Ye−Cy+Mg
B=Mg−Ye+Cy
G=Ye+Cy−Mg
Ye−Cy
Ye+Cy
ye[7..0]
cy[7..0]
mg[7..0]
green[7..0]
4−green_ratio[2..0]
If(IN[10]=1) then
OUT[7..0]=0xFF;
else
OUT[7..0]=IN[9..2];
end if;
If(IN[9]=1) then
OUT[7..0]=0x00;
else
If(IN[8]=1) then
OUT[7..0]=0xFF;
else
OUT[7..0]=IN[7..0];
end if;

FIG. 101

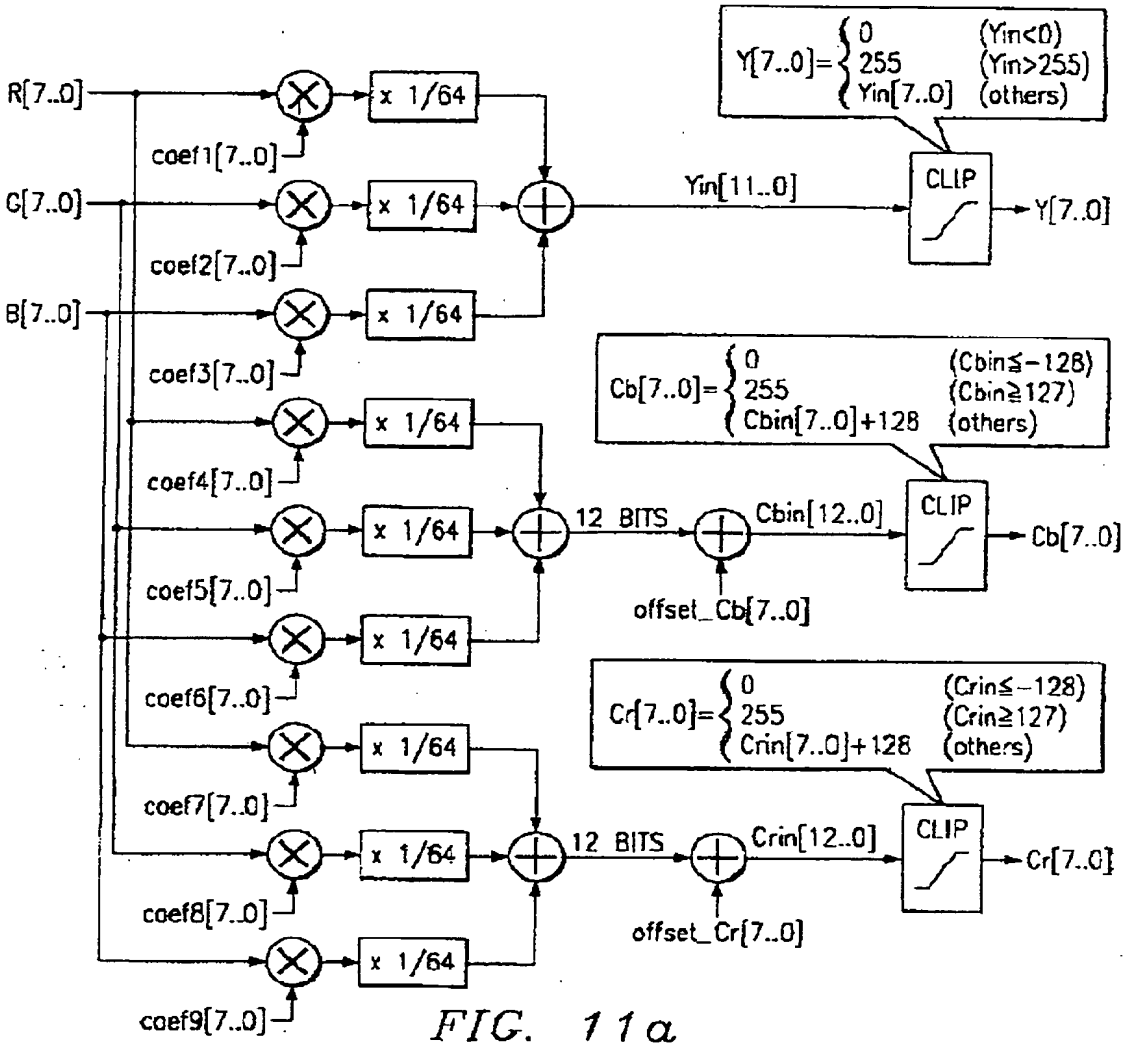


FIG. 11a

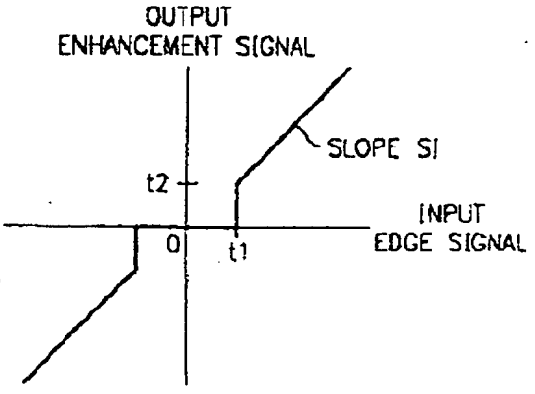
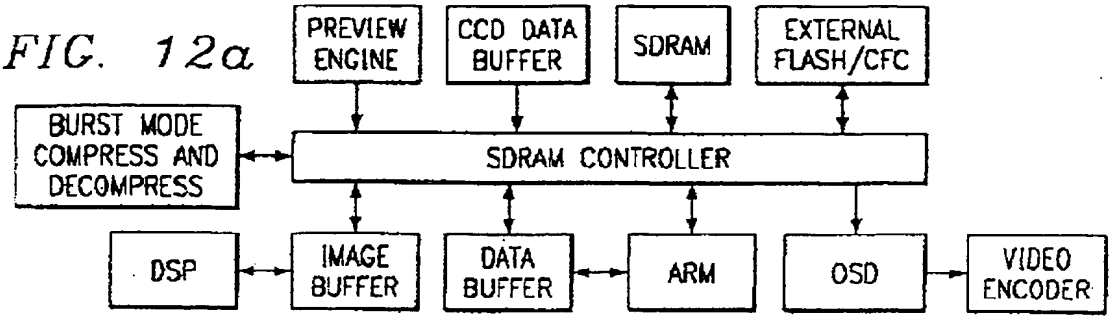


FIG. 11b

FIG. 12a



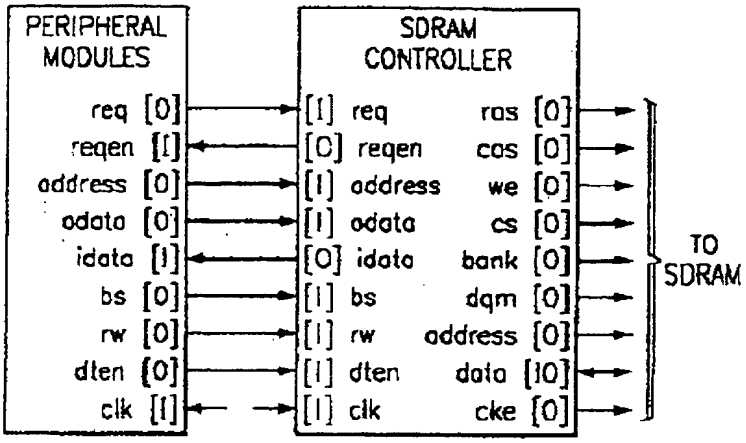


FIG. 12b

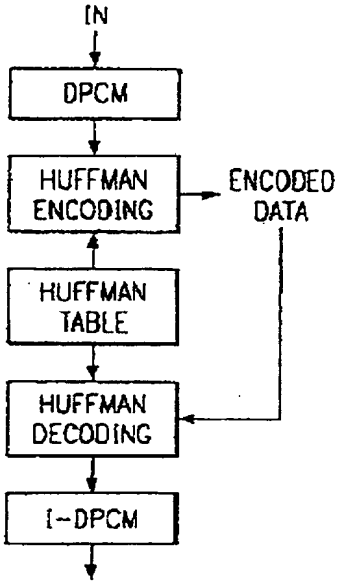


FIG. 13b

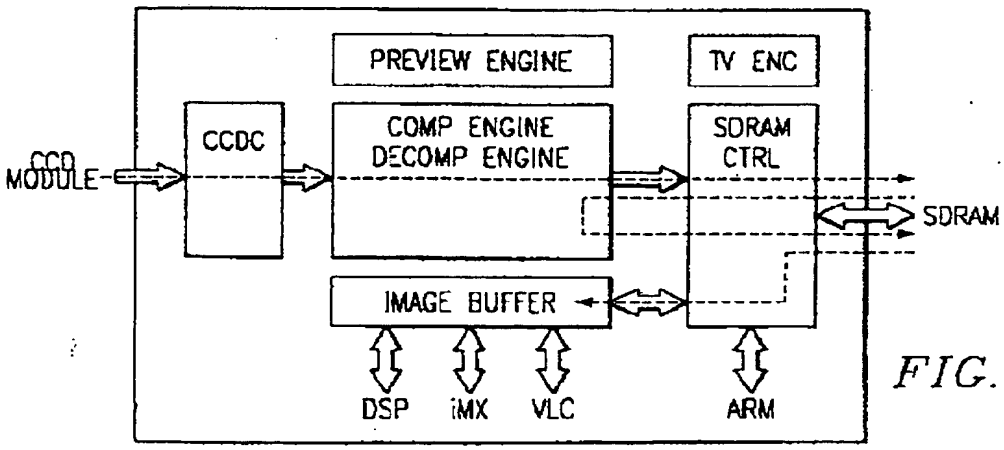


FIG. 13a

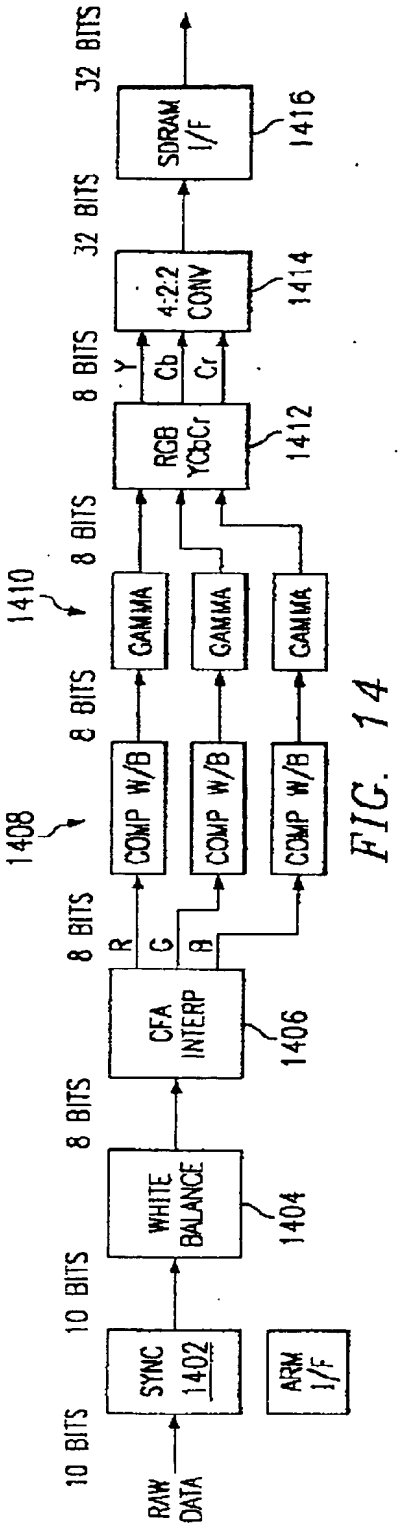

10 BITS
RAW DATA
SYNC 1402
10 BITS
WHITE BALANCE
1404
8 BITS
CFA INTERP
1406
8 BITS
R
G
B
1408
COMP W/B
COMP W/B
COMP W/B
8 BITS
1410
GAMMA
GAMMA
GAMMA
8 BITS
RGB YCbCr
1412
8 BITS
Y
Cb
Cr
32 BITS
4:2:2 CONV
1414
32 BITS
SDRAM I/F
1416
ARM I/F

FIG. 14

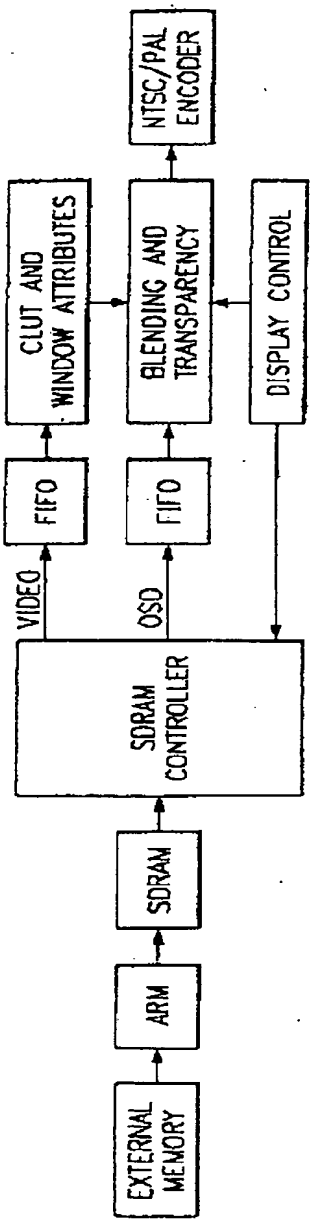

EXTERNAL MEMORY
ARM
SDRAM
SDRAM CONTROLLER
VIDEO
OSD
FIFO
FIFO
CLUT AND WINDOW ATTRIBUTES
BLENDING AND TRANSPARENCY
DISPLAY CONTROL
NTSC/PAL ENCODER

FIG. 15

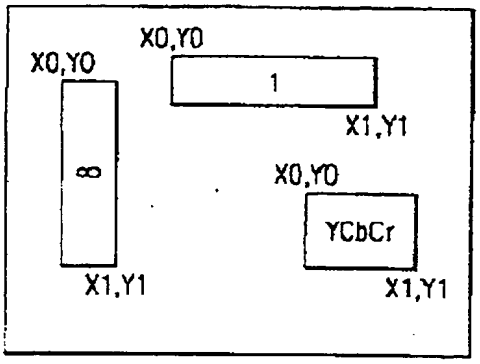


FIG. 16

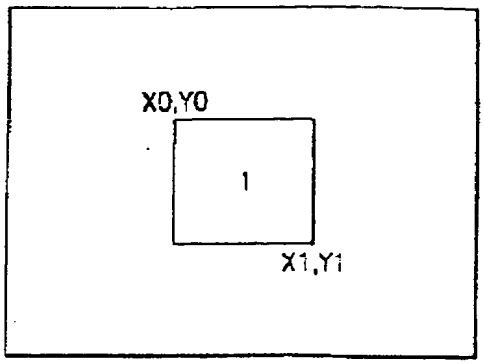


FIG. 17

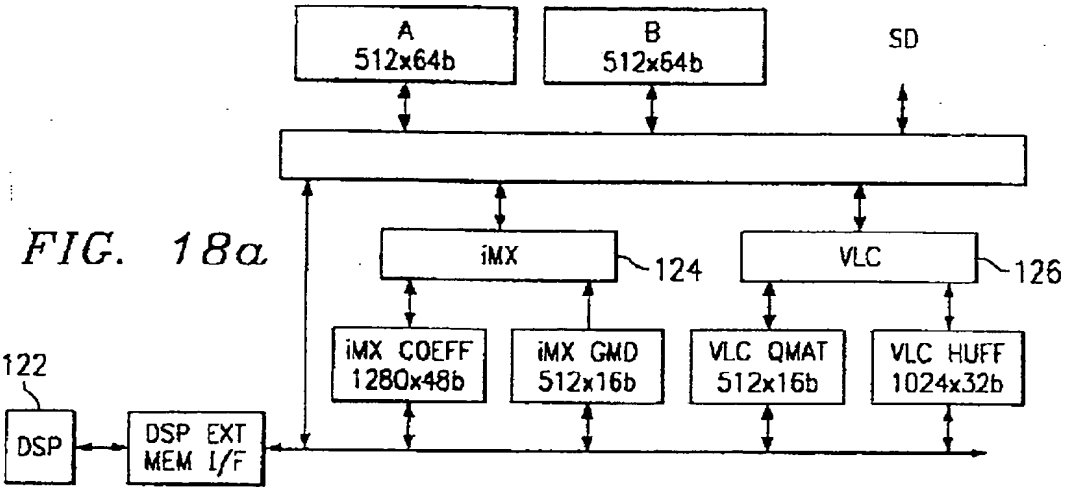


FIG. 18a

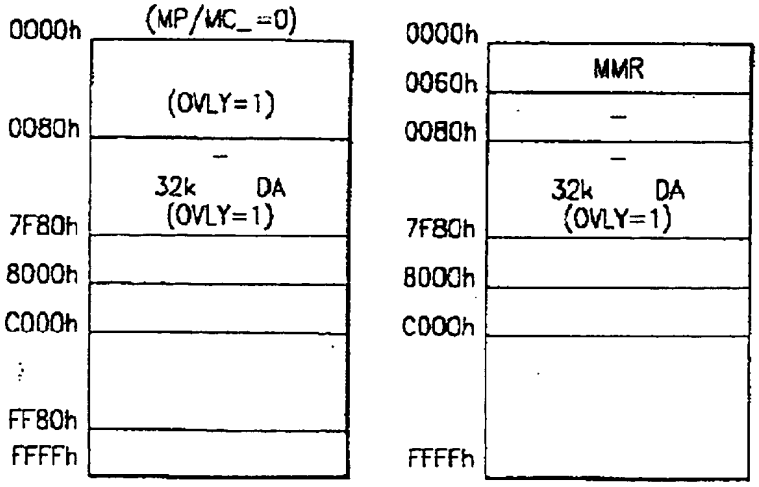


FIG. 18b

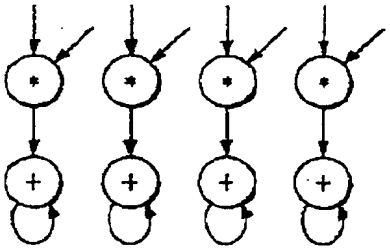


FIG. 19

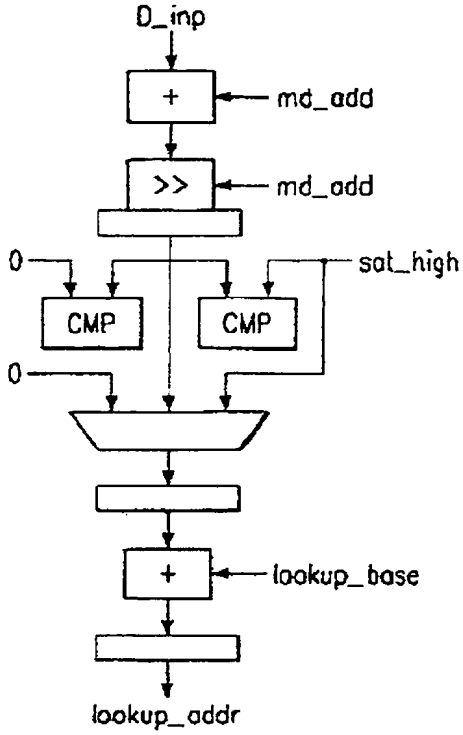


FIG. 21

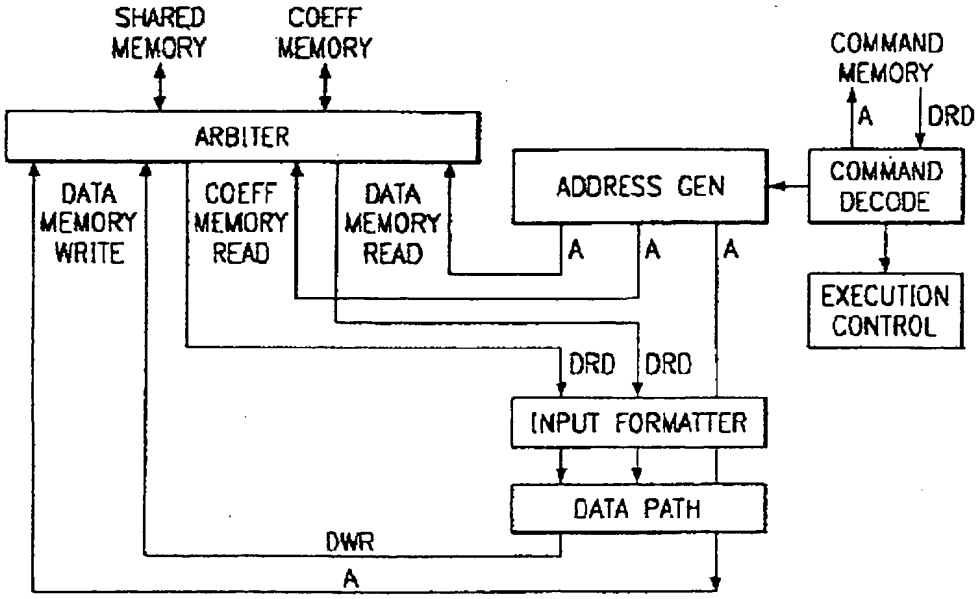


FIG. 20

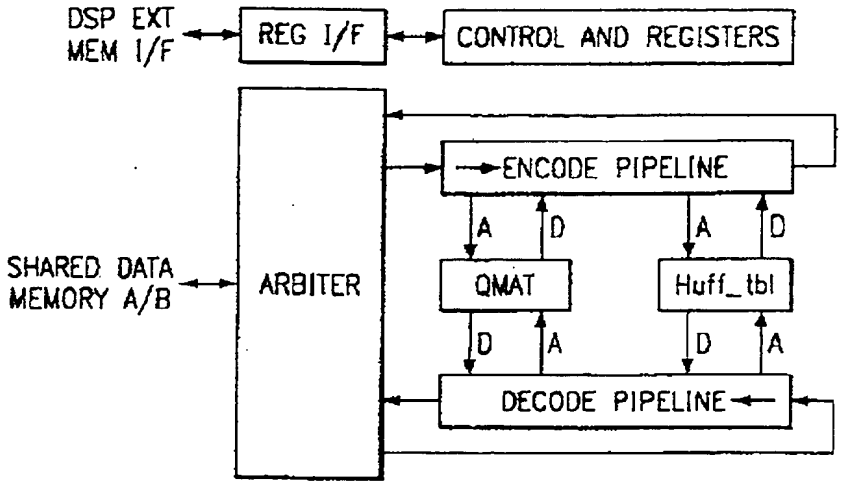


FIG. 22

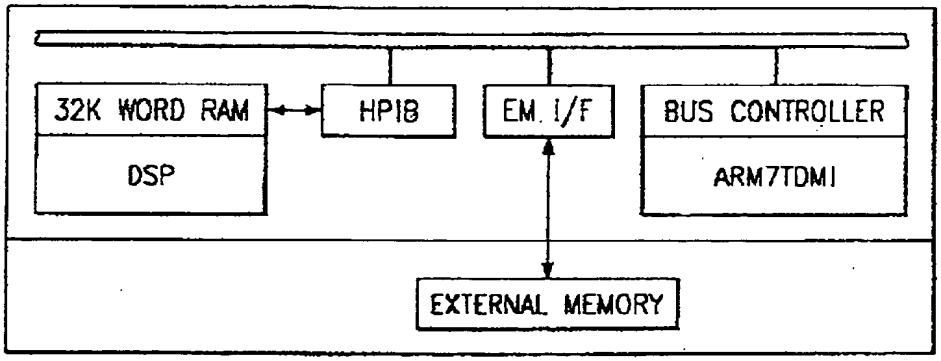


FIG. 23a

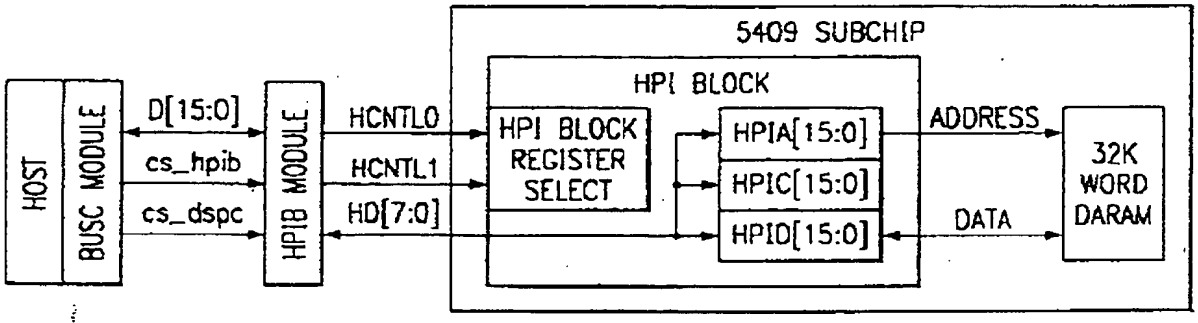


FIG. 23b

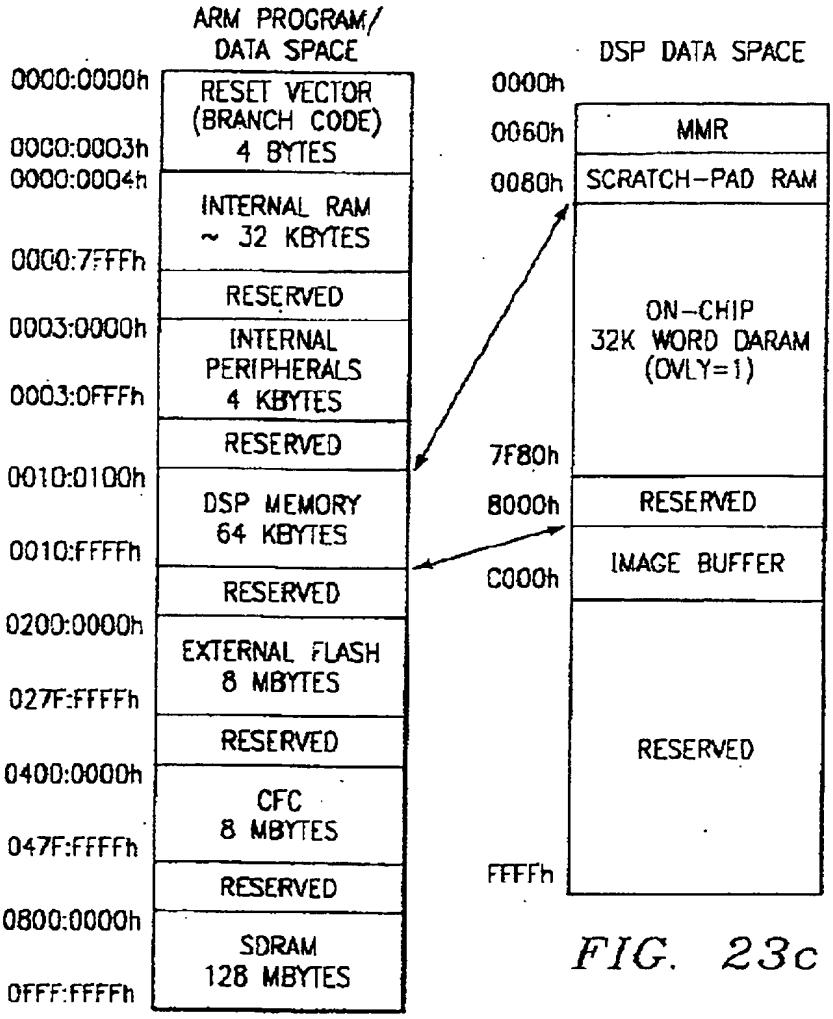


FIG. 23c

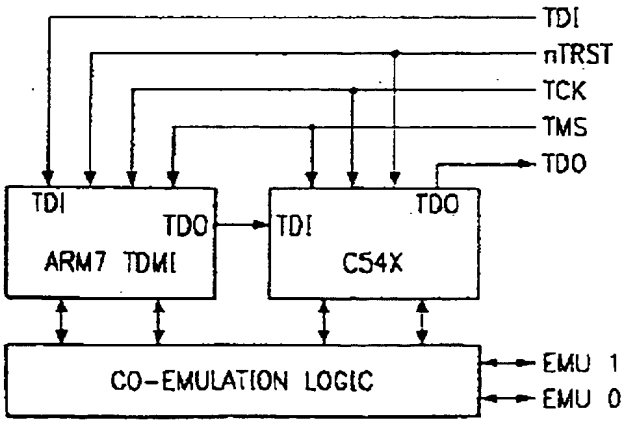


FIG. 24

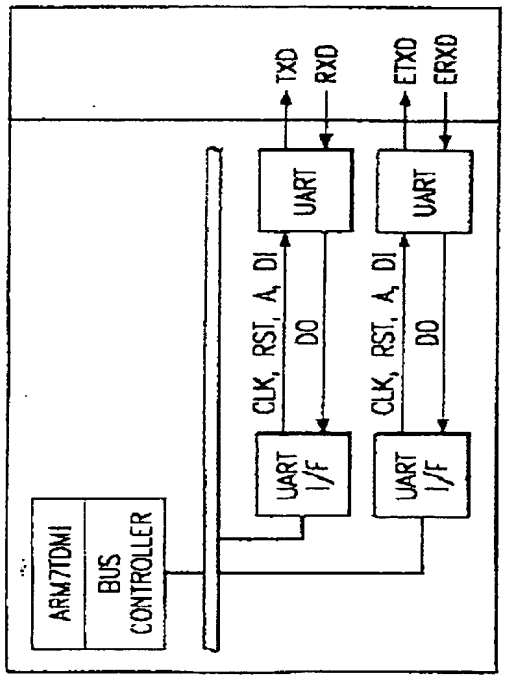


FIG. 25

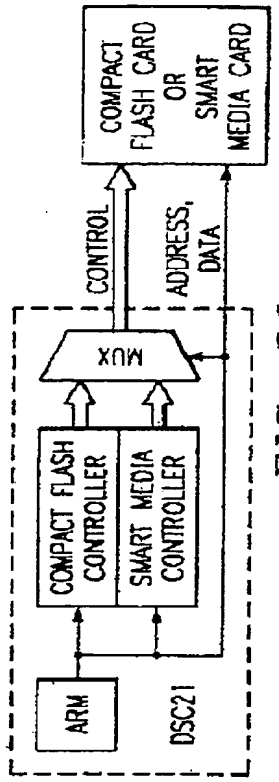


FIG. 26

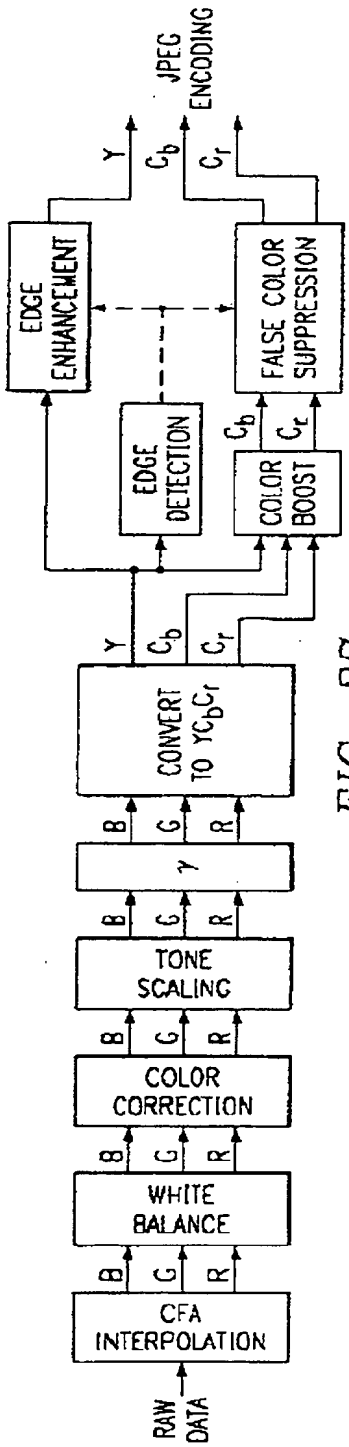


FIG. 27

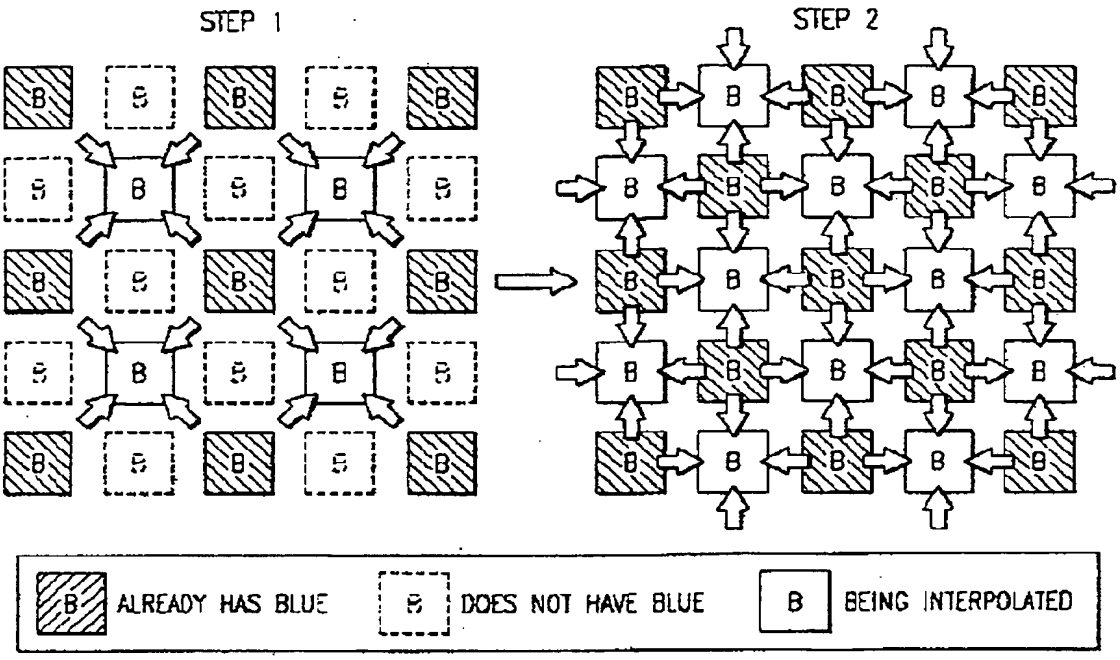

STEP 1
STEP 2
B
ALREADY HAS BLUE
DOES NOT HAVE BLUE
BEING INTERPOLATED

FIG. 28

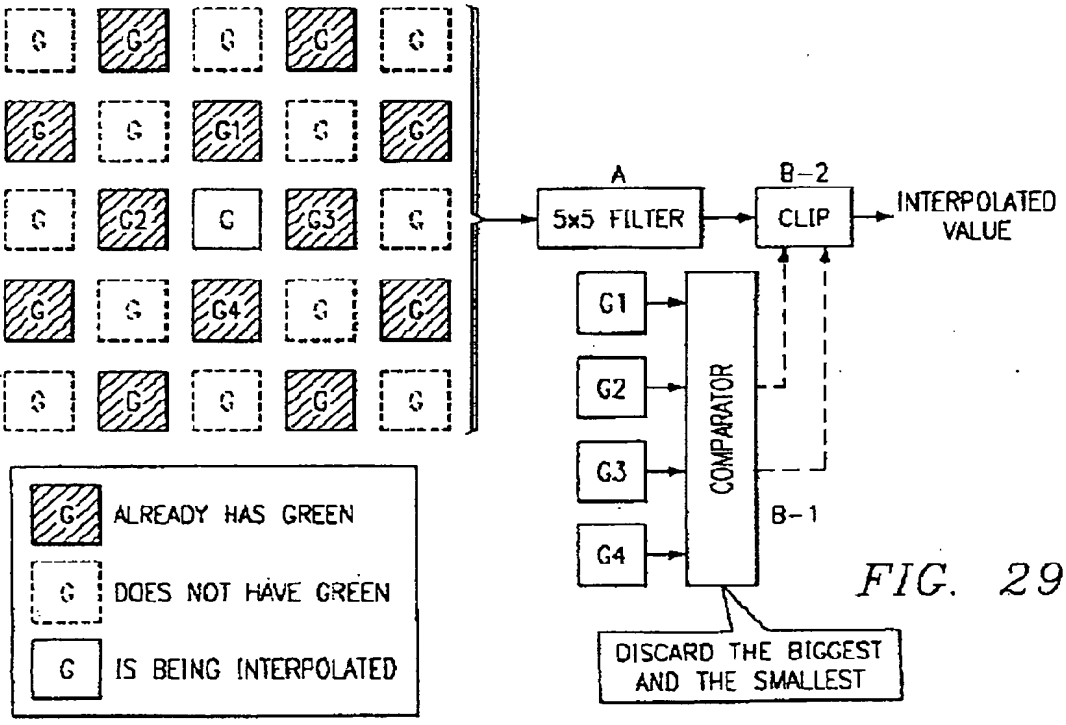

G
G1
G2
G3
G4
ALREADY HAS GREEN
DOES NOT HAVE GREEN
IS BEING INTERPOLATED
A
5x5 FILTER
B-2
CLIP
INTERPOLATED VALUE
COMPARATOR
B-1
DISCARD THE BIGGEST AND THE SMALLEST

FIG. 29

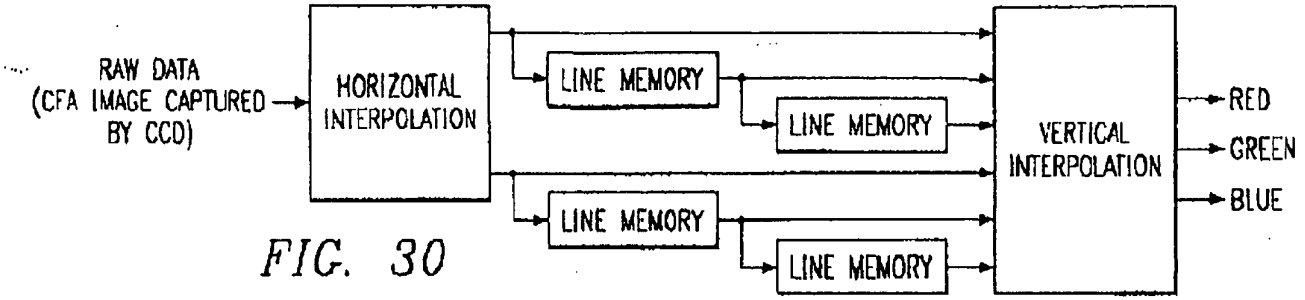


FIG. 30

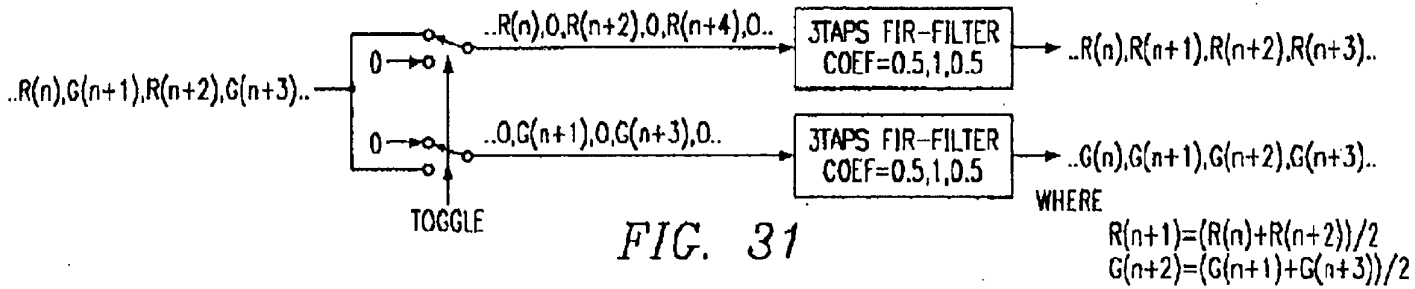


FIG. 31

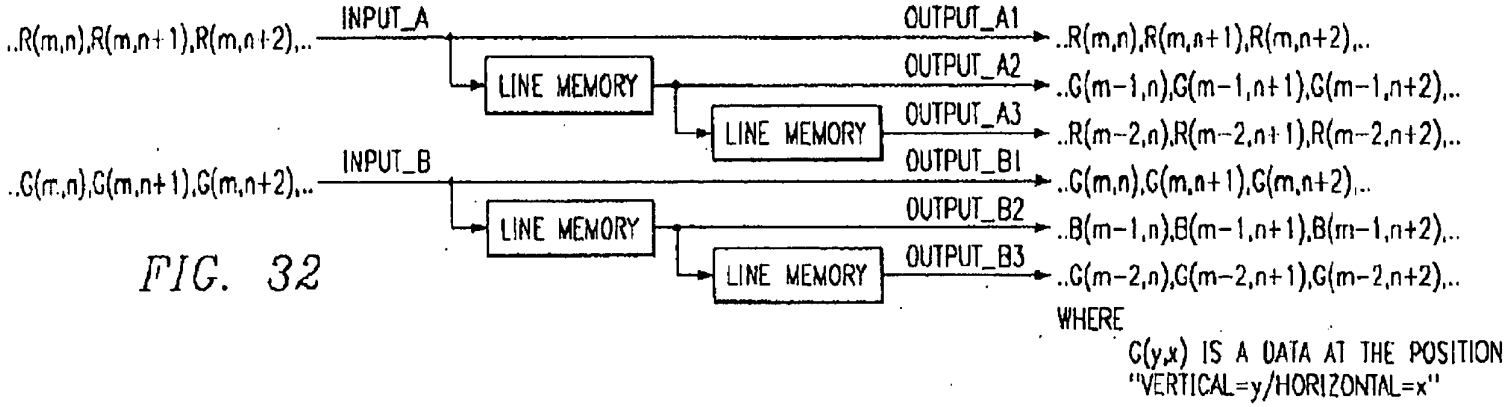


FIG. 32

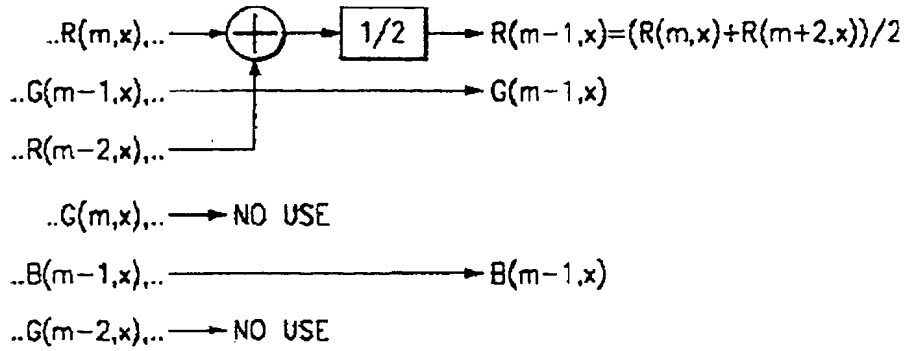


FIG. 33a

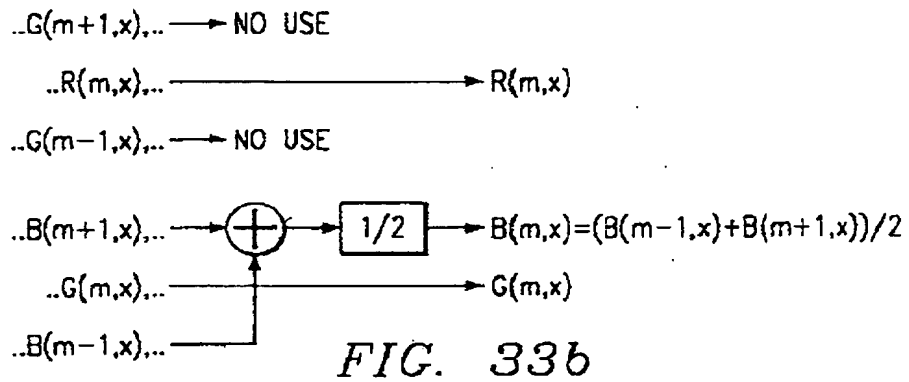


FIG. 33b

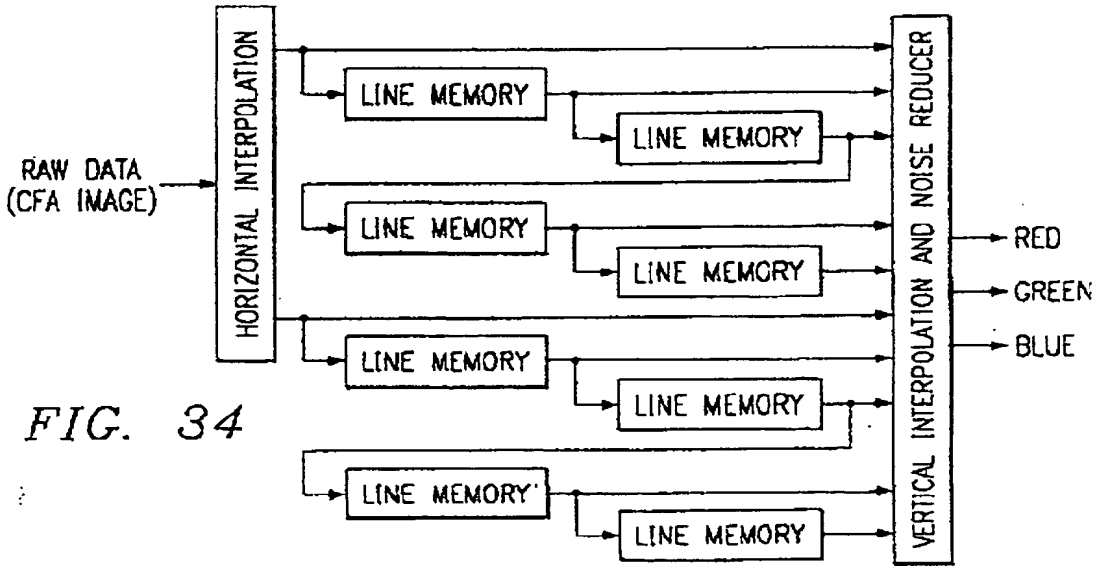


FIG. 34

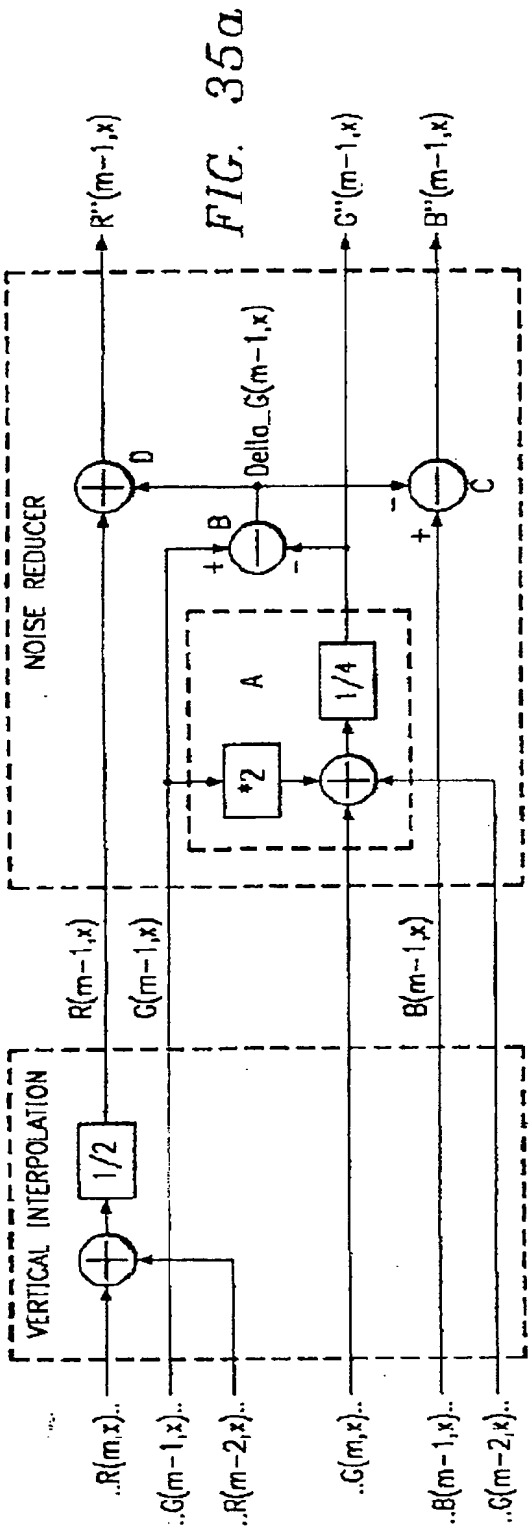


FIG. 35a

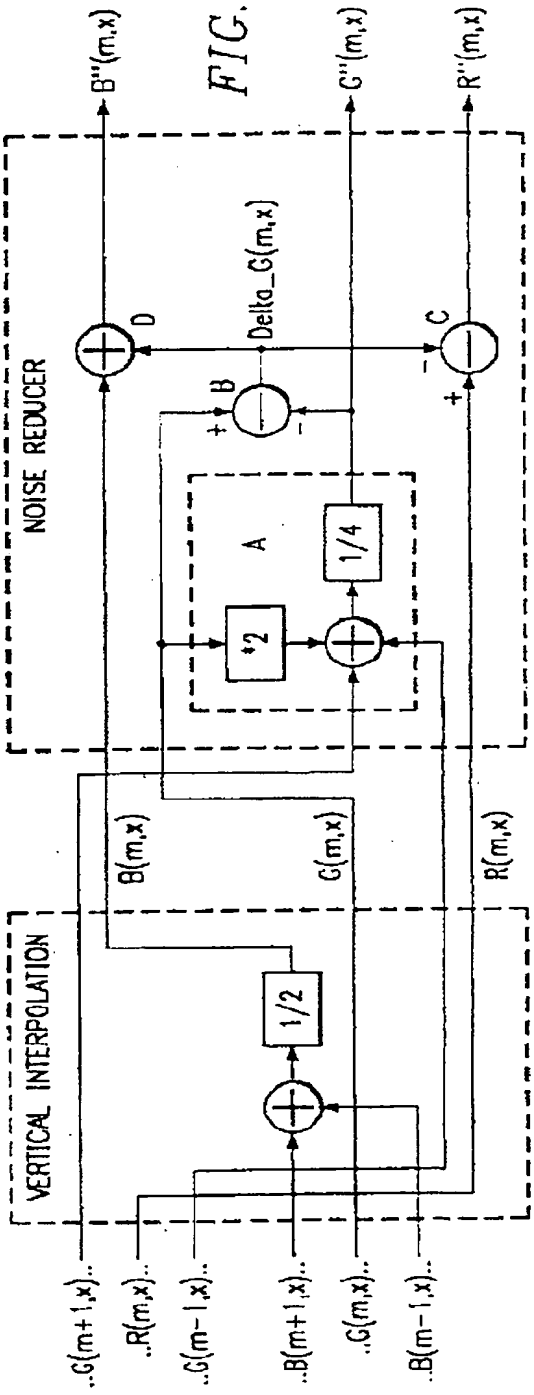


FIG. 35b

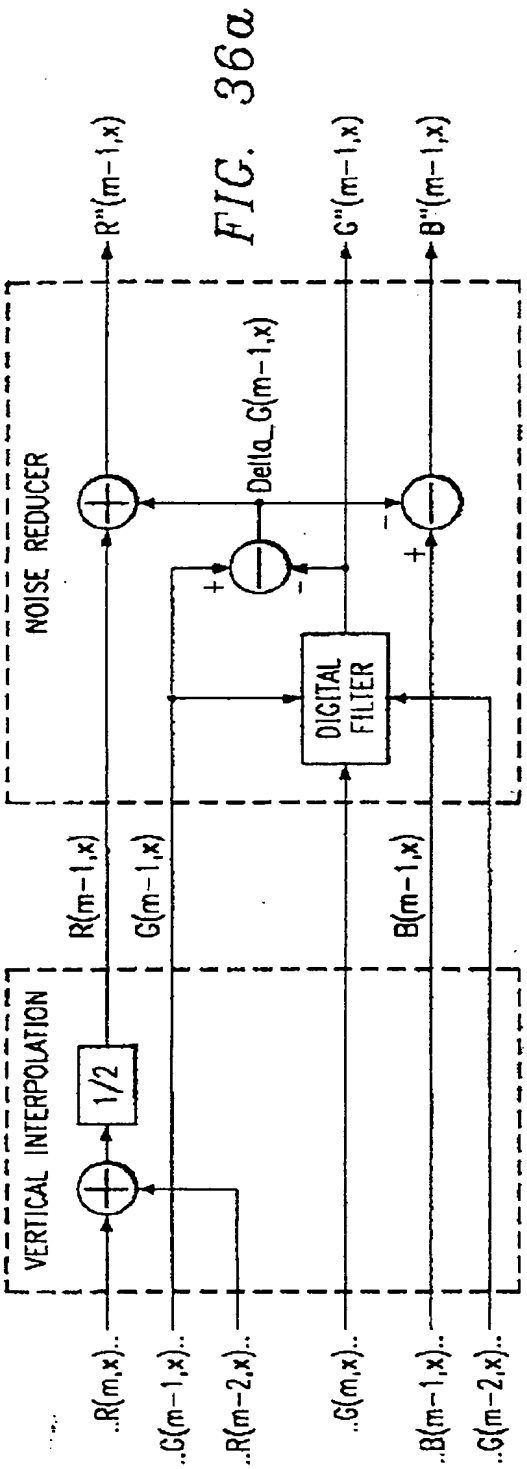


FIG. 36a

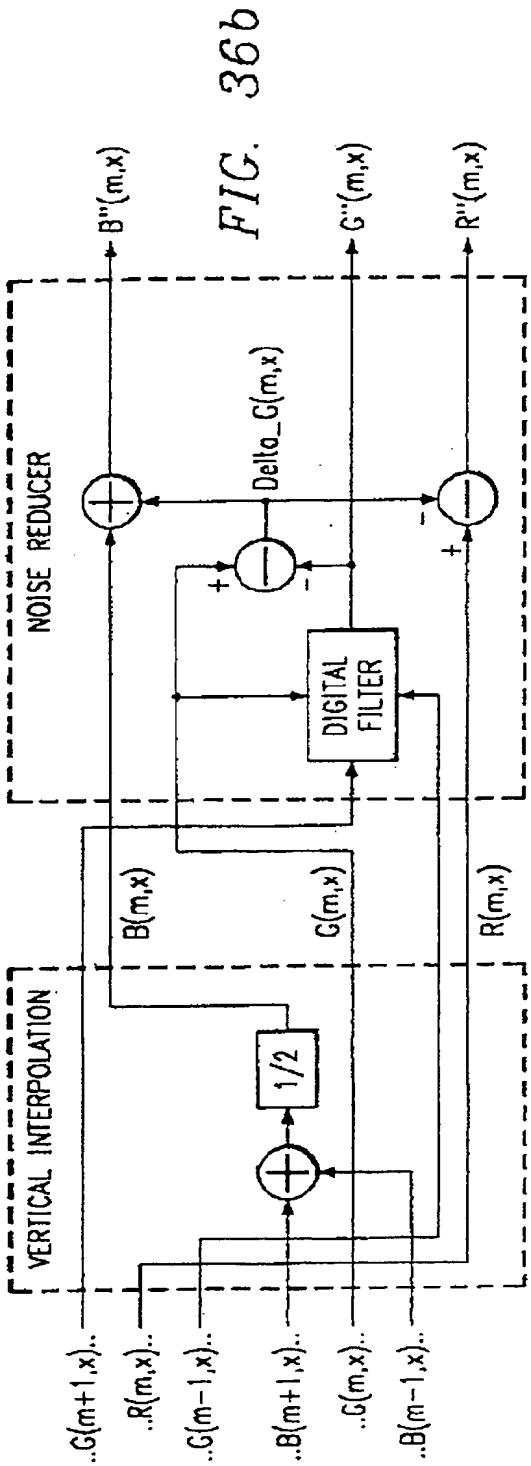


FIG. 36b

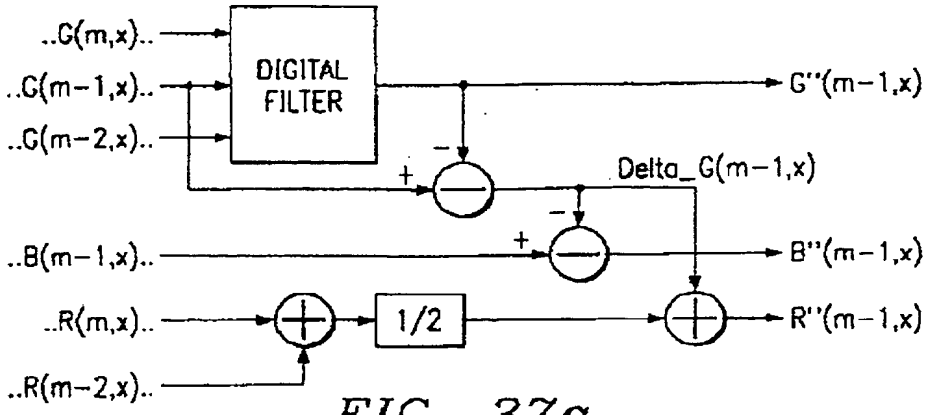


FIG. 37a

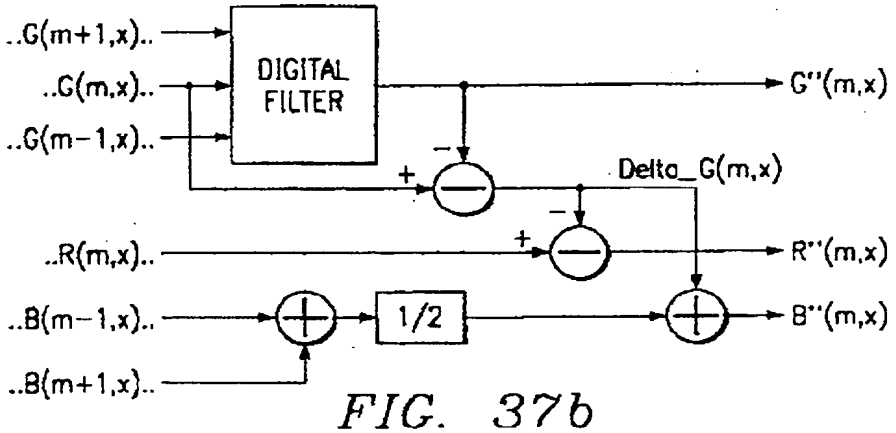


FIG. 37b

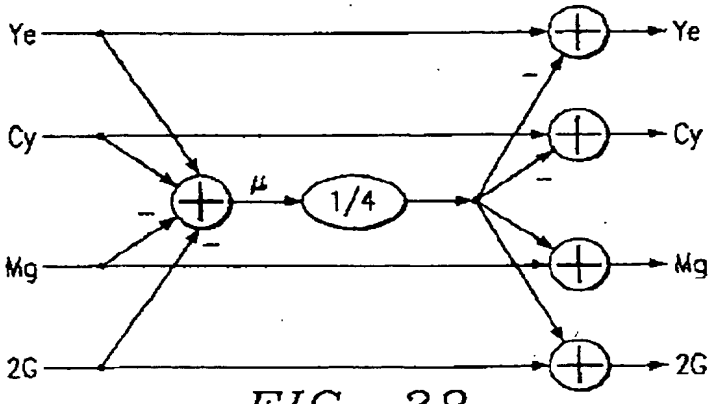


FIG. 38

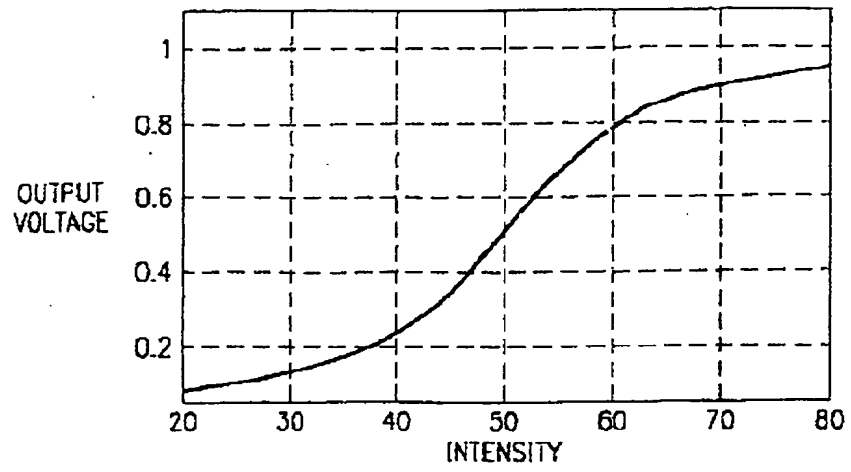


FIG. 39a

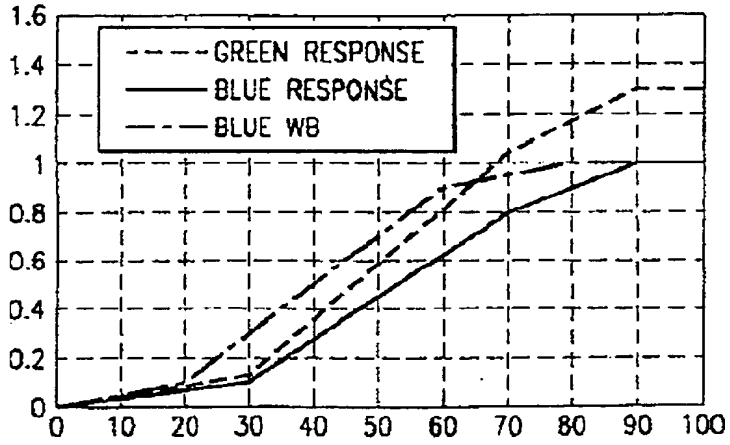


FIG. 39b

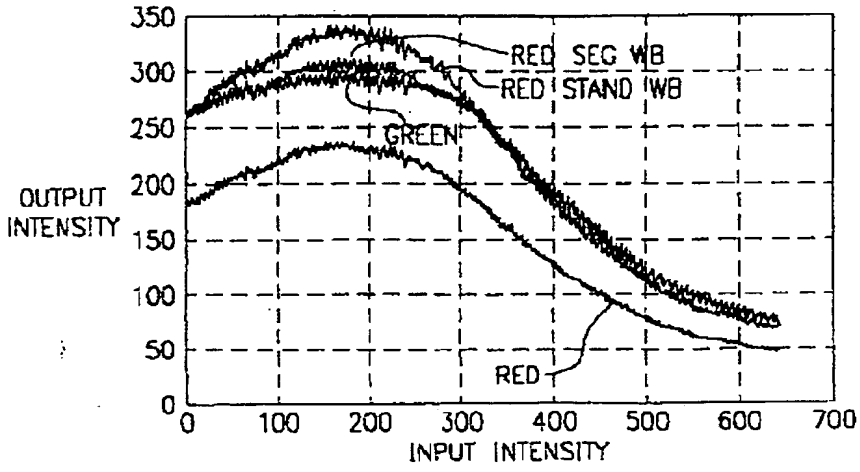


FIG. 40

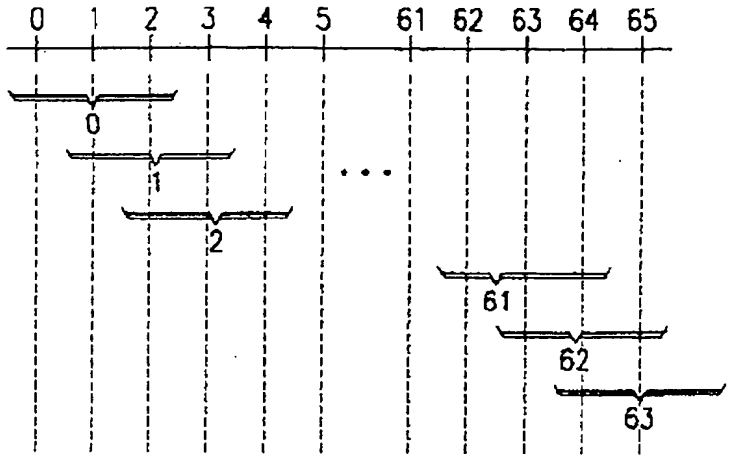


FIG. 41a

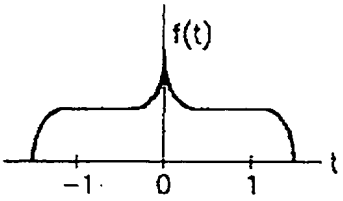


FIG. 41b

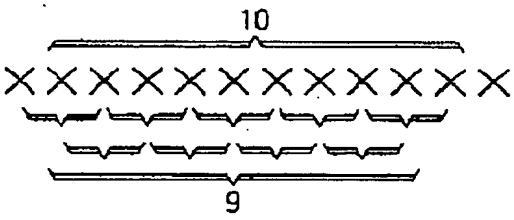


FIG. 42a

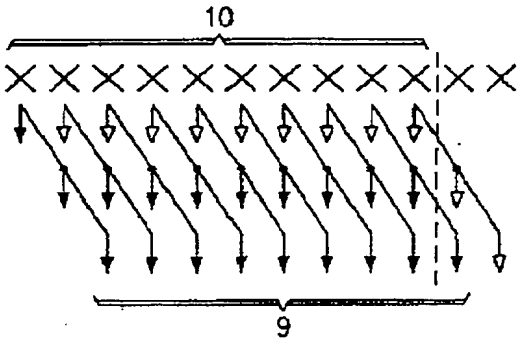


FIG. 42b

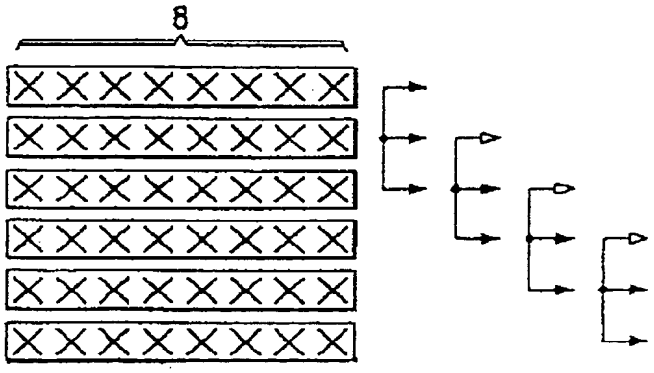


FIG. 42e

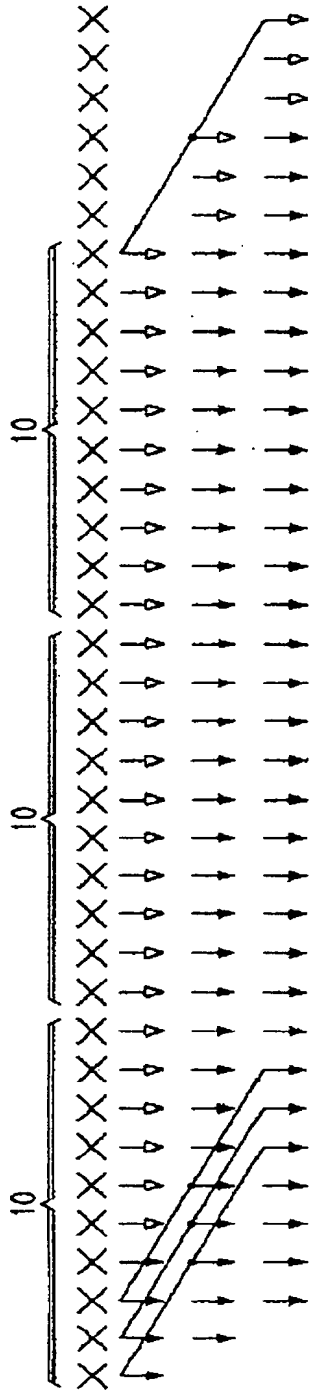


FIG. 42c

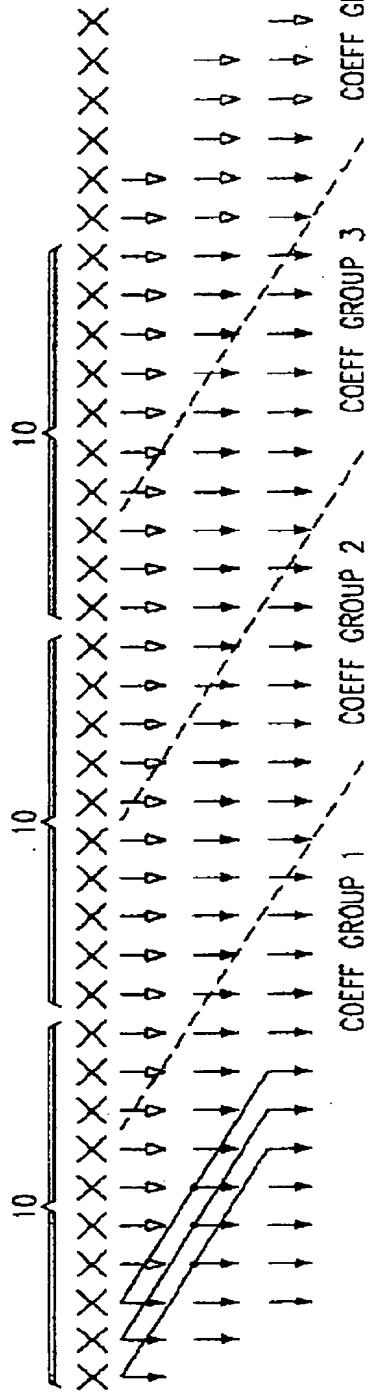


FIG. 42d

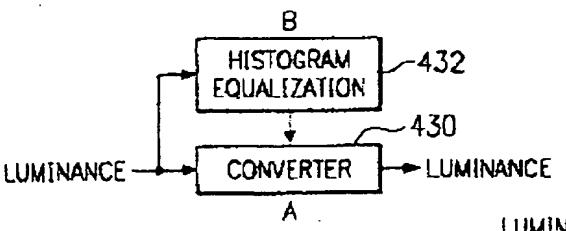


FIG. 43

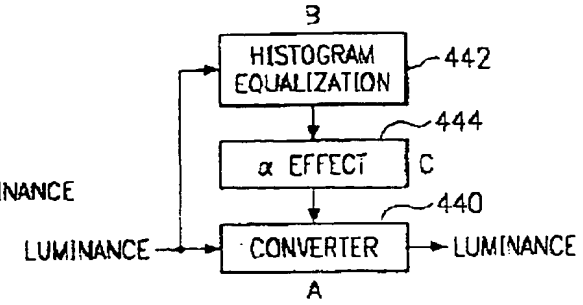


FIG. 44

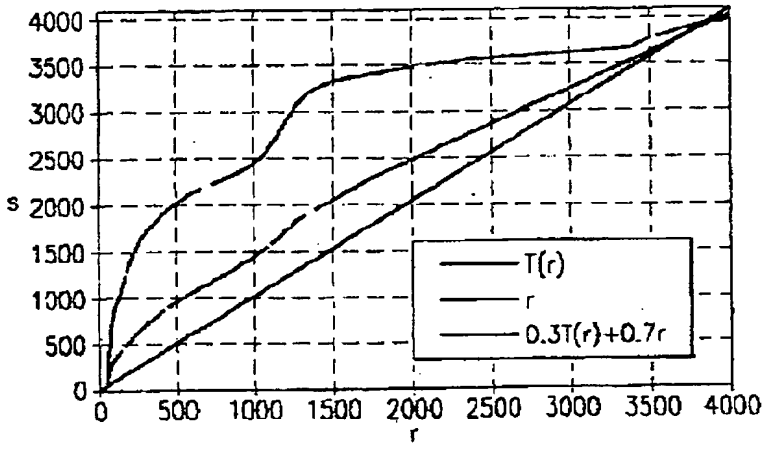


FIG. 45

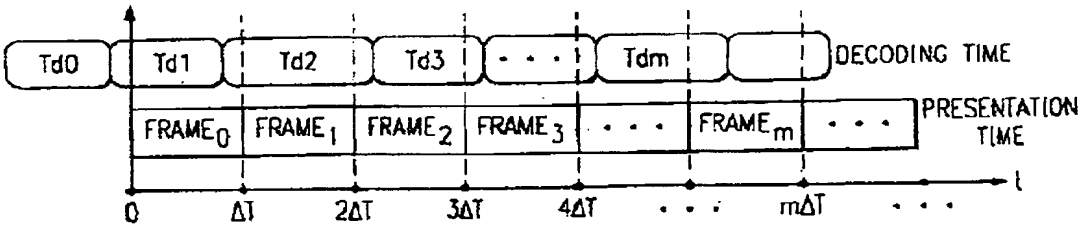


FIG. 46a

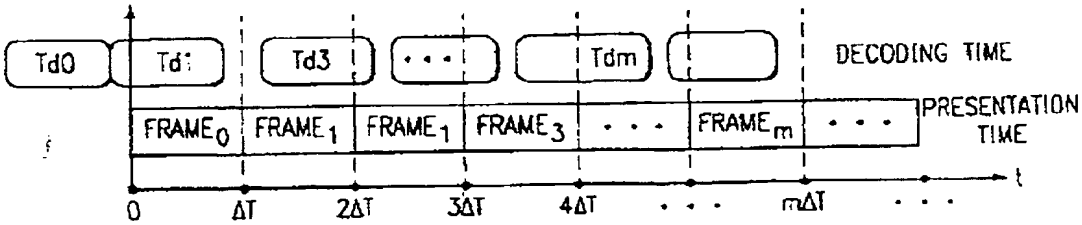


FIG. 46b

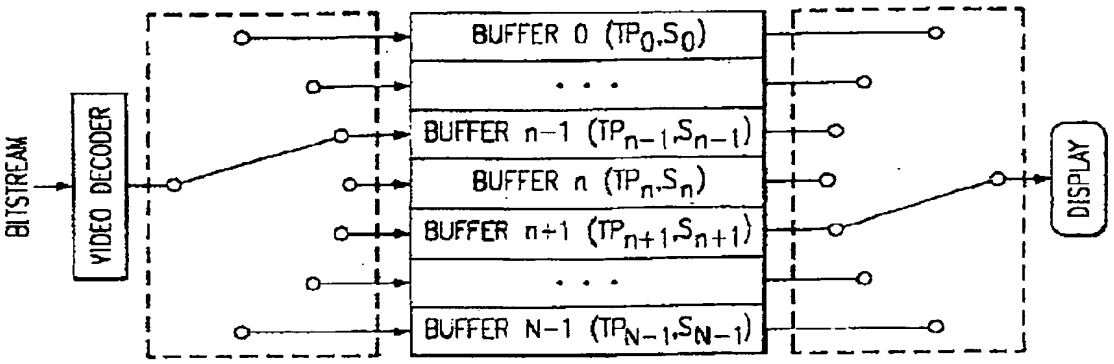


FIG. 47

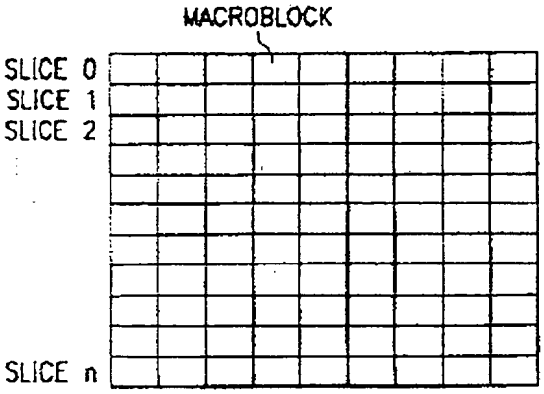


FIG. 49

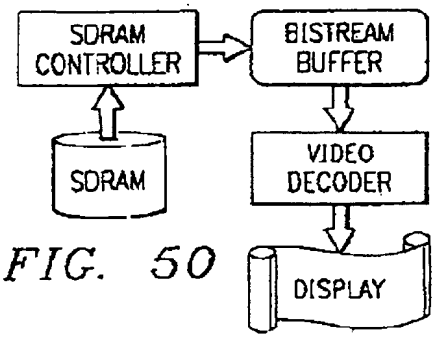


FIG. 50

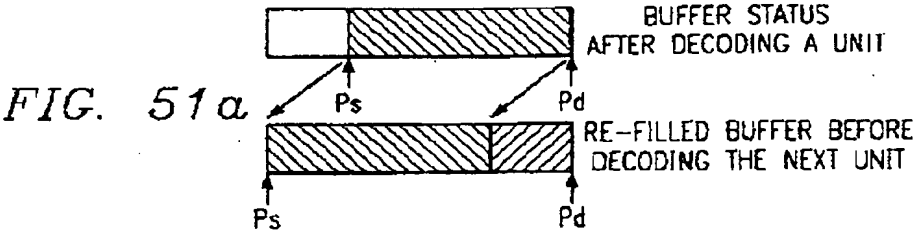


FIG. 51a

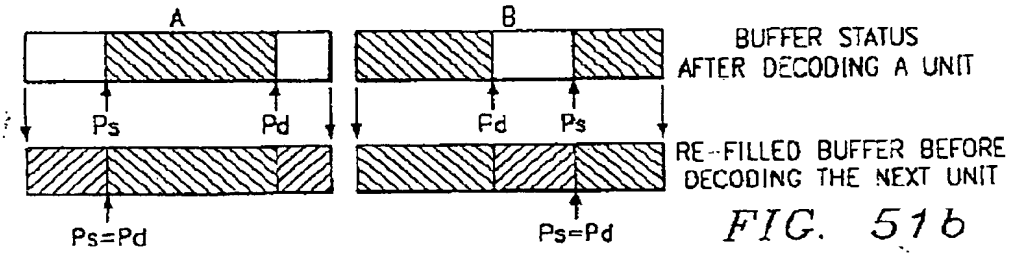


FIG. 51b

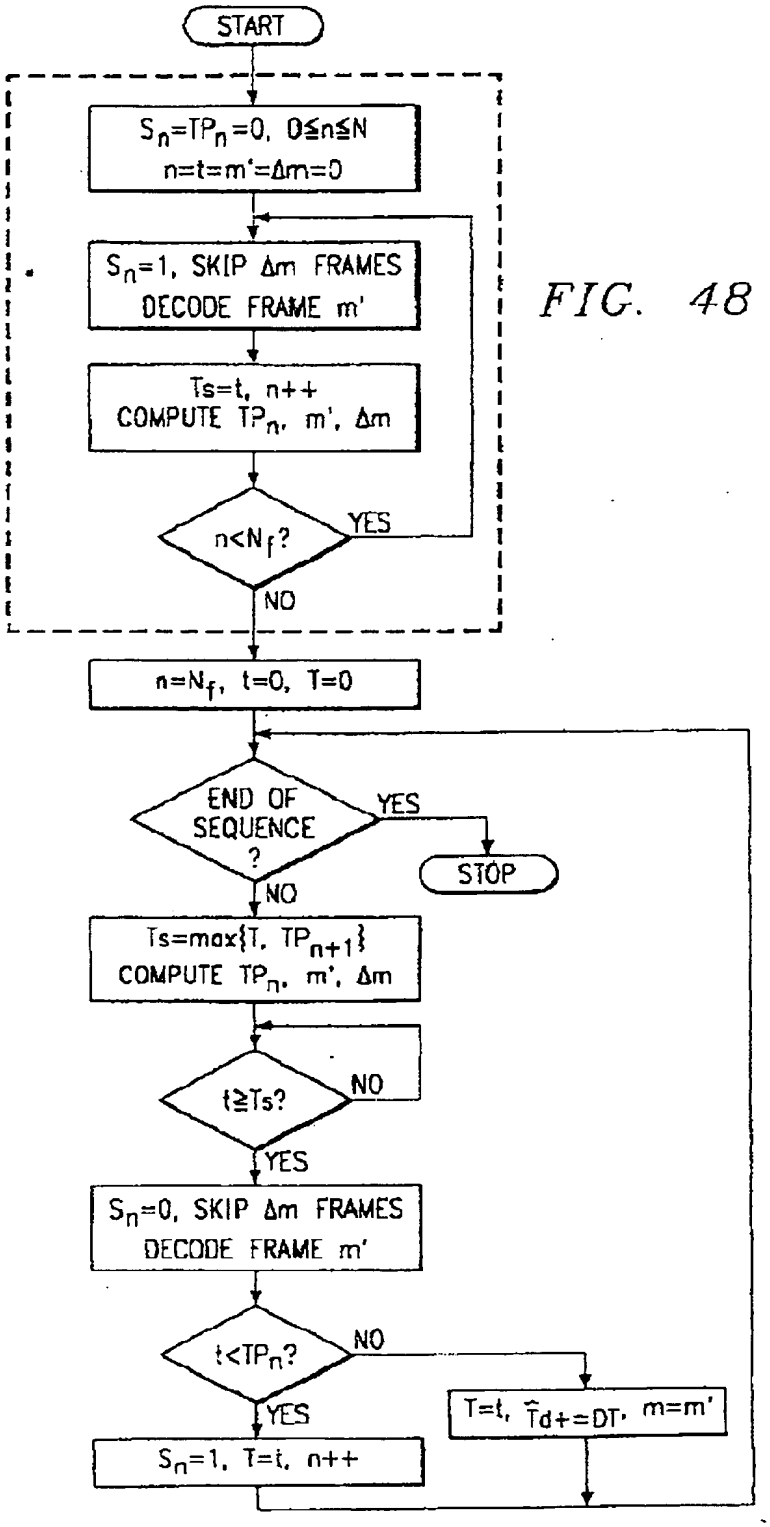

START
$S_n=TP_n=0$, $0 \leq n \leq N$
$n=t=m'=\Delta m=0$
$S_n=1$, SKIP Δm FRAMES
DECODE FRAME m'
$Ts=t$, $n++$
COMPUTE TP_n, m', Δm
$n<N_f$?
YES
NO
$n=N_f$, $t=0$, $T=0$
END OF SEQUENCE ?
YES
STOP
NO
$Ts=max\{T, TP_{n+1}\}$
COMPUTE TP_n, m', Δm
$t \geq Ts$?
NO
YES
$S_n=0$, SKIP Δm FRAMES
DECODE FRAME m'
$t<TP_n$?
NO
YES
$T=t$, $\hat{T}_d+=DT$, $m=m'$
$S_n=1$, $T=t$, $n++$

FIG. 48

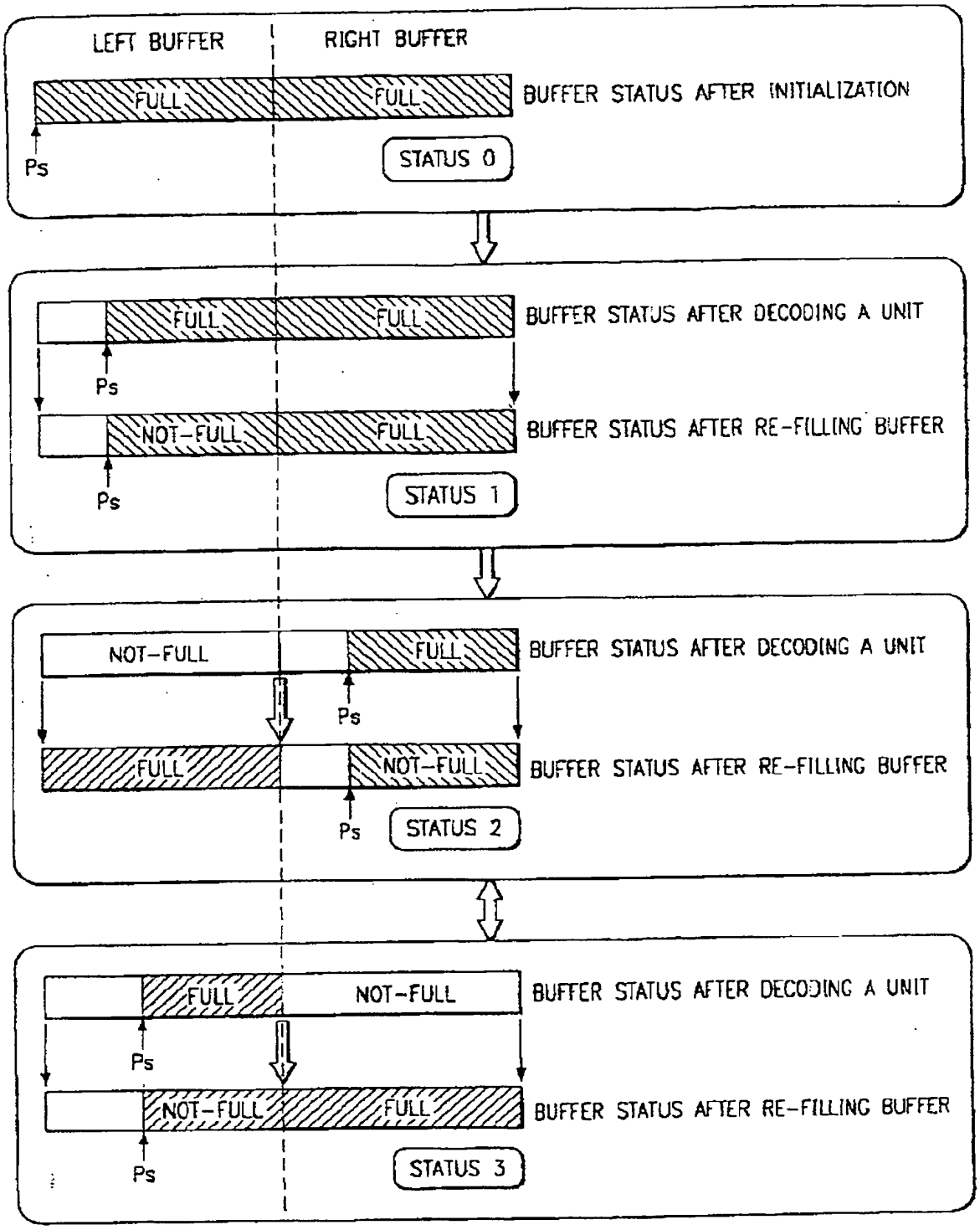

LEFT BUFFER
RIGHT BUFFER
FULL
FULL
BUFFER STATUS AFTER INITIALIZATION
Ps
STATUS 0
FULL
FULL
BUFFER STATUS AFTER DECODING A UNIT
Ps
NOT-FULL
FULL
BUFFER STATUS AFTER RE-FILLING BUFFER
Ps
STATUS 1
NOT-FULL
FULL
BUFFER STATUS AFTER DECODING A UNIT
Ps
FULL
NOT-FULL
BUFFER STATUS AFTER RE-FILLING BUFFER
Ps
STATUS 2
FULL
NOT-FULL
BUFFER STATUS AFTER DECODING A UNIT
Ps
NOT-FULL
FULL
BUFFER STATUS AFTER RE-FILLING BUFFER
Ps
STATUS 3

FIG. 52

1 Abstract

Digital Still Camera (DSC) includes separate preview engine, burst mode compression/decompression engine, image pipeline, CCD plus CCD controller, and memory plus memory controller. ARM microprocessor and DSP share control. Color filter array interpolation by use of green high-frequency added to red and blue plus interpolation.

2 Representative Drawing

Fig. 1a